3D Gadgets for Business Process Visualization —a case study—

Bastiaan Schönhage[1,2], Alex van Ballegooij[1,3] and Anton Eliëns[1,3]

1 - Vrije Universiteit
Department of Mathematics and Computer Science
De Boelelaan 1081, 1081 HV Amsterdam, The Netherlands
email: bastiaan, alex, eliens@cs.vu.nl

2 - ASZ Research & Development, Gak Group NL
P.O. Box 8300, 1005 CA Amsterdam, The Netherlands

3 - CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

Abstract

Business visualization is becoming increasingly important, since managers recognize the power of human visual intuition in information-rich decision tasks. Nevertheless, despite its promises, 3D visualizations are far less common than one would expect.

In this paper, we describe a case study where we took a 2D visualization of a business process as a starting point, for which we subsequently provided a 3D visualization. We introduce a small set of 3D visualization gadgets and associated behaviors, implemented in Java3D, that proved to be relatively complete for our case.

For each of these gadgets and behaviors, we will discuss requirements and design trade-offs. The case study, which concerns an actual business process of the largest social security provider in the Netherlands, illustrates the usability of our gadgets and their associated behaviors, which include brushing, grouping, and (drill down) manipulation.

CR Categories and Subject Descriptors: I.3.2 [Computer Graphics]: Graphics Systems - Distributed / network graphics; I.3.8 [Computer Graphics]: Applications; D.2.12 [Software Engineering]: Interoperability - Distributed objects

Additional Keywords: Java3D, Information Visualization, Gadgets, Business Process Visualization

1 Introduction

In the DIVA project we deploy visualization components which are called visualization **gadgets**. These gadgets can present information in a two- or three-dimensional fashion, dependent on the goal and target group of the visualization. In the past, we have created three-dimensional gadgets based on VRML and Corba as described in [6]. To overcome problems signaled there, such as the limited interactive possibilities, we decided to experiment with the new high-level 3D API offered by Sun, Java3D [9]. In order to validate requirements and design tradeoffs in a realistic setting, we recreated an existing 2D visualization of a particular business process in 3D, using our Java3D gadgets.

The case we used to experiment with the designed Java3D gadgets concerns the visualization of business information at Gak Netherlands in the domain of social security. Section 2 describes the context of this case study and shows an example of the 2D prototype that was the result of a previous case study. In addition, we will discuss why we made the transition to 3D visualization of business processes. After that, Section 3 presents the reusable collection of Java3D gadgets we needed to make the transition from 2D to 3D visualization in this case study. The set contains both behavior and visualization components. The application of these gadgets to visualize business information is described using the case study at Asz / Gak in Section 4. Finally, in Section 5 we will end with conclusions.

2 Managing Business Processes at Gak Netherlands

Gak is the largest social security provider of the Netherlands. One of the core businesses of Gak NL is processing applications for benefits. For example, when people become unable to work they will have to go to the Gak to apply for a benefit. After that, their application goes through a number of stages, such as medical inspection and ability assessment. This process is the business process that we will use as the source of our information visualization later.

However, before that, we will first explain why we choose to visualize business processes in this domain. In the near future, social security will no longer be state-owned and Gak will loose its current monopoly. To obtain a strong position in the market the Gak company wants to improve the production time and efficiency of their production process without losing the quality of their products. The delivered products (in this case products are benefit appliances) are qualitatively good but the production process from appliance of the worker to deliverance (approval or disapproval) takes too

long. The norm production time for an appliance is in most of the cases 13 weeks, but due to unknown problems in the production process the production often exceeds this norm. The Gak company wants to find the production bottleneck and the cause of it.

To assist the managers controlling the processes, we have built a prototype visualization system [7]. The system was built in two phases. First, we concentrated on the current problems of the managers to indicate the bottlenecks in the business process. In this case, our data source was a database containing measurements of the time applications needed at several stages of the process. In the second phase of the project, we created a simulation of the business process and used the previously designed visualizations to display the results.

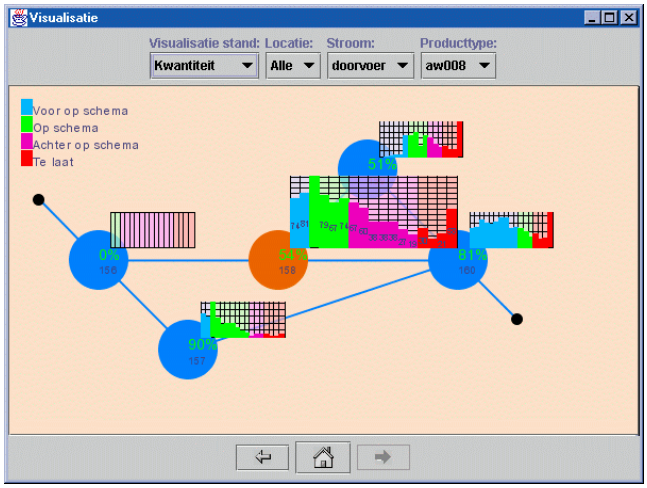

Figure 1: 2D visualization of the throughput of the business process

Figure 1 contains a screenshot of one of the visualizations created to discover bottlenecks in the current production. The combination of the process structure and related colored histograms helps managers to quickly find possible piles of work in the current throughput of the process. Other visualizations were created to give an overview of the past in order to search for trends in the available data. By means of simulation, managers are able to asses the results of possible interventions such as adding people to the workforce. In summary, the prototype uses the same visualizations to view past, present and future.

2.1 Moving from 2D to 3D Visualization

3D graphics and its application in serious information visualization applications is a matter of dispute. On the one hand, its advocates promote the usage of 3D because of its close relation with human's three-dimensional intuition. Additionally, 3D visualizations can contain more information at once and are therefore better suitable of presenting large sets of data. Opponents of 3D, on the other hand, present the following main difficulties of using 3D on a computer: the input devices fall short to control the 3D space and therefore distract users from their primary task [2]. Additionally, 3D applications are often considered as toy applications because they look nice without adding relevant new features.

Experiments with two and three-dimensional visualizations indicate that it is very difficult to compare 2D and 3D directly. A striking phenomenon is the fact that people with more computer experience significantly gain better scores with 3D interfaces than novices. Sebrechts *et al* therefore rightly conclude that *3D visualization cannot be adequately evaluated using only short-term studies of novice users* [8].

According to our opinion, 3D can definitely add value as soon as both technical problems, such as bad input devices and slow machines, and human problems —people will have to get acquainted to 3D as they had to get acquainted to graphical interfaces— are solved. Until then we will have to keep on building better 3D hardware and software.

3 A Reusable Collection of Visualization Gadgets in Java3D

Java3D is the 3D application program interface (API) of the Java language. It is used to create platform-independent 3D applications that can be used over the internet. Additionally, Java3D can read and display VRML files and combine VRML scenes with Java3D contents.

Although Java3D offers some high-level building blocks, such as built-in primitives (sphere, cone, etcetera) and behaviors for interaction, it still requires a lot of programming effort to create a simple visualization. To fill in this gap, we have created a collection of reusable visualization gadgets based on top of Java3D.

The set consist of two types of primitives. First, we have the behaviors which usually do not present themselves graphically, but merely exist to add interaction to a scene graph. The behaviors we will discuss here are brushing and manipulation. The second class of components we have created, the gadgets or visualization primitives, can reveal information by means of 3D graphical representations. The gadgets we will describe here are the cone tree, the histogram and the graph.

3.1 Behaviors

The DIVA Java3D collection currently contains five different types of behaviors. Two of them (brushing and modify behavior) are discussed in somewhat more detail below. The current behaviors are:

- **BrushingBehavior** reveals extra information about the object that the input device is currently pointing at.
- **KeyBehavior** is a generic behavior class to move objects through the scene according to key presses. It is often used to move the camera viewpoint.
- **MenuBehavior** displays a context-sensitive 3D menu when the users selects an object.
- **ModifyBehavior** adds many possible manipulations of a Java3D object to a single mouse button. Supported examples include rotation, translation, scaling and iconification of groups.
- **TranslateBehavior** translates 3D objects in such a way that it moves along the screen's x-axis and y-axis (instead of the object's x and y-axis).

3.1.1 BrushingBehavior

Brushing allows to retrieve more detailed information about parts of a visualization without changing the visualization itself. By moving a pointing device over a particular component in the visualization extra information appears on top of the selected object. The advantage of brushing is that more detailed information can be retrieved quickly without replacing the current visualization. A simple mouse movement is enough to reveal, for example, the numbers on which the visualization is based.

The goal of the brushing behavior is to allow a program to easily add information in the form of brushing to an existing scene graph. For the brushing behavior we have three requirements:

- It must be possible to add brushing information to an existing scene graph.
- It must be possible to dynamically change the information that is associated with brushable objects.
- The brushing behavior must be relatively efficient, because it is expected to be used often and should therefore not pose too much of a burden on the system.

The brushing behavior we developed satisfies the defined requirements. One can add it to a scene graph after it has been created and dynamically change both what 3D objects are brushable and what information is to be associated with them. Figure 2 shows a simple test scene containing a box, pyramid and sphere.

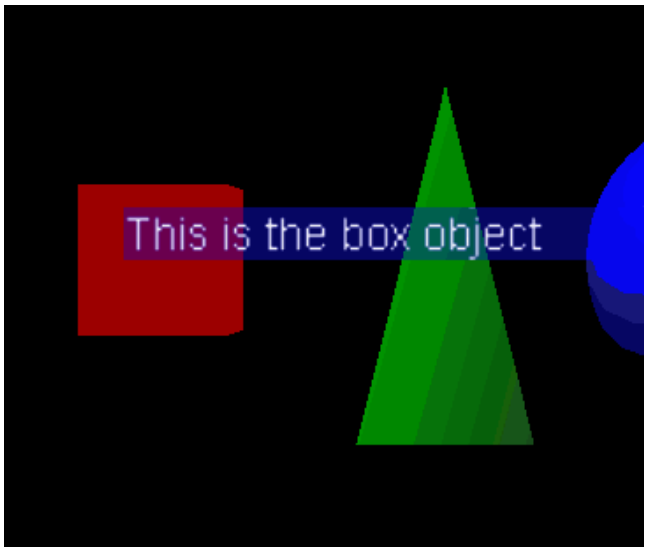

Figure 2: Brushing the red box

We tried different approaches to implementing the behavior. At first we attempted to use ordinary 2D graphics to display information. While this does work, it was apparent that it goes directly against the 3D paradigm and thus reduces the realism of the 3D graphics. What we chose to do instead is to add a 2D plane to the 3D scene on which we use a texture map to display the intended textual information.

3.1.2 ModifyBehavior

An important way of interacting with a visualization is the ability to manipulate the objects present in the 3D scene. The standard Java3D behaviors support this by means of a combination of mouse and key presses. A problem with these standard behaviors is that the controls are not intuitive and use all three mouse buttons. Besides, many users do not even have 3 mouse buttons, but are limited to two or even a single button. On top of that the standard implementations manipulate objects relatively to their local co-ordinate system. If one moves an object in the x-direction using such a behavior, the object moves along its own x-axis. Depending on the viewpoint and the object's orientation, this might be a completely different direction than the mouse moved. In short the standard manipulation behaviors are not what we want.

The requirements for the modify behavior are:

- The behavior must be selectively applicable. In other words, the program must be able to indicate what objects the user can and cannot manipulate.
- The behavior must use as few different controls as possible and preferably allow the user to do all kinds of manipulations using only a single mouse button.
- The behavior must be extendable, it should be possible to add additional manipulation methods.
- It would be best to indicate to the user what object (or group of objects) he or she is manipulating.
- The basic rotation, translation and scaling manipulations should work intuitively and relatively to the viewpoint. For instance, moving an object to the left along the x-axis should always result in the object moving to the left along the x-axis of the screen, regardless of the orientation of the object and the current viewpoint.

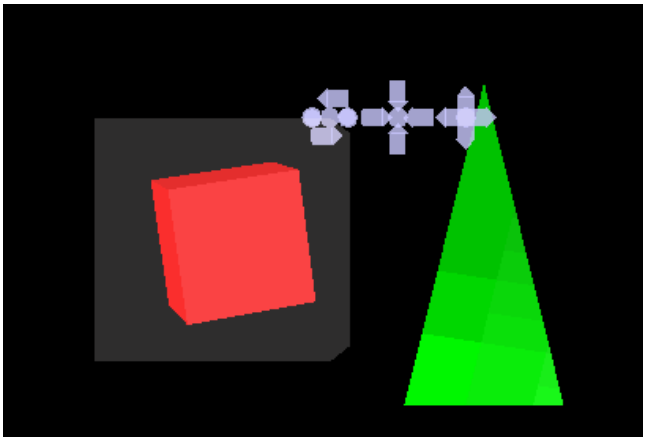

Figure 3: The red box' modify behavior has been activated

The modify behavior allows users to pick an object. The selected object is surrounded by a transparent box to indicate selection. Users can manipulate it by pressing one of the buttons that the behavior shows. Figure 3 shows a simple scene containing a box, pyramid and sphere. The box object has been selected and rotated a little. The figure clearly shows that the red box is selected, because of the transparent white box drawn around it and the three little buttons indicate the three possible actions: rotation, translation and scaling. As we will see when discussing the graph gadget, additional buttons can easily be defined and added.

3.2 Visualization gadgets

Currently, the collection of visualization primitives in the DIVA package comprises three gadgets: the cone tree, the histogram and the graph gadget. This set proved to be relatively complete for our current purposes of visualizing business processes. Each of these gadgets will be discussed in somewhat more detail below.

3.2.1 The Cone Tree

The cone tree was developed at Xerox Parc and has since then become one of the best known examples of 3D visualization [3]. The idea behind the conetree is that the 3D representation of a tree structure makes optimal use of the screen space and thus enables the visualization of much larger structures than the traditional 2D approach [1].

The requirements for our implementation of the cone tree are as follows:

- As the cone tree is meant to visualize large data sets, an important requirement is that the implementation must be efficient. It must be able to display several thousands of nodes without a problem.
- The implementation must be able to display dynamic trees. Changes in the underlying data-structure must be reflected as changed in the displayed cone tree.
- The user must be able to navigate through the data set by manipulating the layout of the tree. In the case of the cone tree this is realized by allowing the user to rotate each of the cones, to view it from the direction that contains the data the user is interested in.
- Users must be able to select nodes in the tree. Additionally, the gadget must be able to change the constellation of the branches in such a way that the selected path is rotated to the front.
- The tree must be able to highlight paths through the tree to show the path from the root of the tree to a certain node. This can give the user insight into the structure of the data.

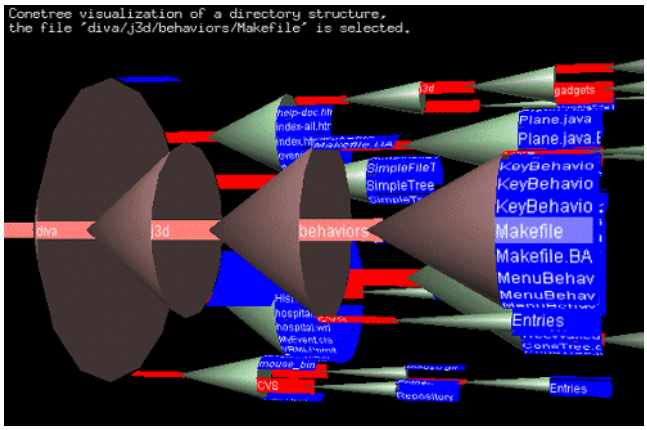

Figure 4: A Cone Tree visualizing a directory structure

To show the usefulness of the cone tree we have chosen to visualize the directory structure of a file system as illustrated in Figure 4. Our implementation was designed with performance as the major consideration. This has proved to be successful. Our latest version of the tree can display about a hundred times as many nodes as our original implementation. The flexibility of the implementation has suffered clearly from this approach though. Addition of new features is extremely difficult without changing a lot of elements in the code. The inflexibility is mainly the result of the fact that the tree structure is not explicitly maintained in the gadget. Only relations that are necessary for the proper operation of the gadget are maintained in (hash) tables, unessential relations have been sacrificed to performance. Also the shapes of leaves and nodes have been fixed to special text planes and cones, again in order to improve performance.

3.2.2 The 3D Histogram

The histogram is probably one of the best known visualization primitives. The advantage of a 3D histogram over a 2D histogram is the fact that the third dimension can be used to show additional information. This way, different data sets, e.g. the monthly sales results of the past year, can much easier be compared.

As requirements for our 3D histogram we came up with the following:

- The dimensions of the histogram must be customizable.
- The histogram must be able to handle an arbitrary number of bars of arbitrary size and shape.
- The text displayed at the axis must be customizable.
- The histogram must be able to change dynamically whenever the input values for the visualization change.

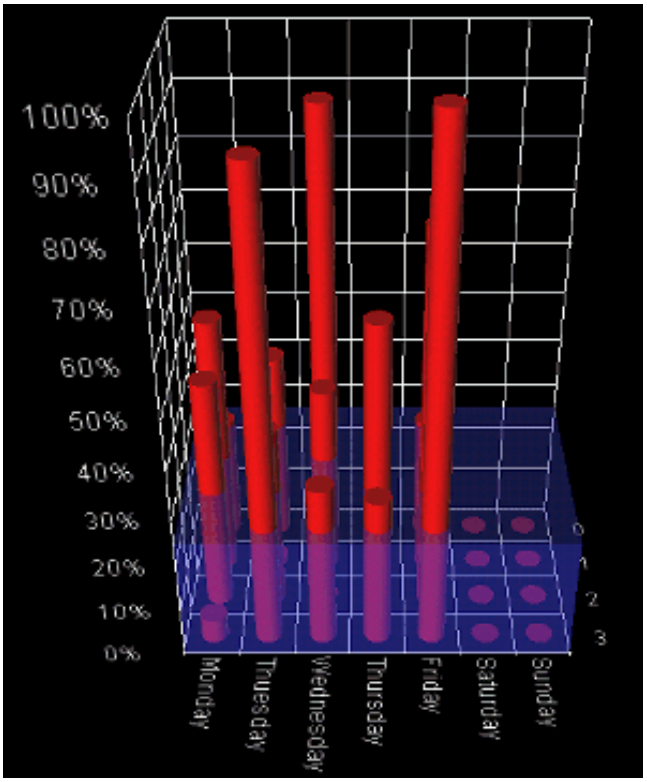

Figure 5: The histogram gadget

Figure 5 shows a screenshot of the 3D histogram in action. Because 2D histograms are often used in (business) visualizations, we have designed the gadget to resemble 'normal' 2D histograms. The 3D implementation, however, adds the possibility to show multiple rows of bars at the same time. In addition to this, the gadget has a so-called water-level, which can be used to indicate to the user what level is to be considered normal. The water-level can be seen in Figure 5 as the transparent blue box filling the lower 25 percent of the histogram gadget.

An important feature is that the histogram works with generic bars. All that a bar object must do for the histogram gadget to be able to use it is implement the Bar interface. This makes it possible to implement specific bars for specific purposes. There are two standard implementations of the Bar interface included in the DIVA package. The first option is a simple bar, which is nothing more than a red cylinder that can grow and shrink. A second implementation is the multi-level bar, which allows for a customizable number of sub bars with customizable colors that can all grow and shrink independently of each other. The histogram gadget uses the brushing behavior to allow users to retrieve additional information about each of the bars.

3.2.3 The Graph

The purpose of the Graph gadget is to visualize graph structures and transitions, in particular dynamic (business) processes. It is capable of representing both the simulation model (a graph) and the elements going through the model of a running simulation. Requirements for the graph gadget include:

- The graph gadget must be able to visualize reasonably large structures.
- The gadget must be able to visualize activities within the graph, e.g. by means of tokens flowing over the edges of the graph.
- The user must be able to change the layout of the graph.
- The user must be able to hide sections of the graph in order to focus on what is important.
- The user must be able to select nodes and provide feedback to the visualization as a whole and the underlying information source (e.g. a simulation).

The graph gadget consists of nodes which are connected by edges. Tokens move from node to node over the edges to illustrate their flow through the visualized model. The gadget is event driven and all elements are uniquely named for identification. Manipulation of separate graph elements is achieved by use of the ModifyBehavior. The BrushingBehavior provides the user with information about nodes and tokens when they are brushed. To allow users to select and (de)iconify groups two new buttons were implemented for the modify behavior. One button allows users to select the group in which the currently selected item is contained. The other allows users to (de)iconify the currently selected group.

Since the graph gadget is meant to visualize simulations, the test-case for the gadget is a simple simulation of a counter-based business process, such as a bank or a fast food restaurant. Figure 6 shows the test simulation. It represents a simple model of a bank, people (represented by tokens) enter the bank and wait in the queue until one of two counters is empty. Whenever a person is waiting in the queue and a counter is empty the first person in the queue leaves the queue and walks over to the free counter where he or she is being served. After this, the person leaves the bank.

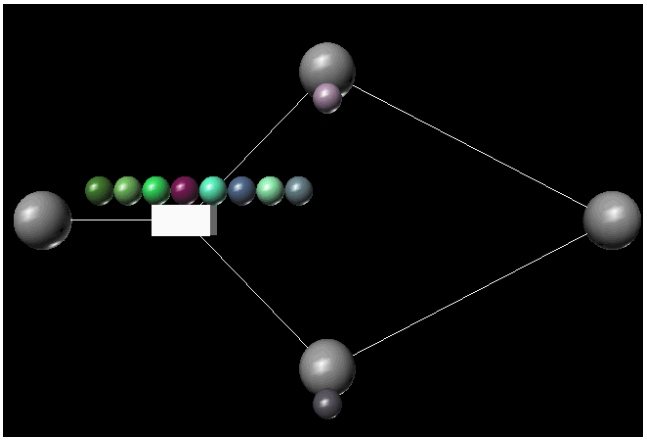

Figure 6: The graph gadget

3.2.4 Additional Gadgets

This set of gadgets can be expanded, more visualization primitives can be thought of that would be useful. The results thus far and the experiences gathered indicate that Java3D is well suited for this kind of work. However, our current purposes were not to create an exhaustive set of 3D visualization primitives. Rather, we wanted to rebuild a 2D visualization application using 3D techniques to evaluate the advantages and disadvantages of the 3D approach to business visualization. The collection of gadgets we have built allows us to experiment with 3D visualization in an actual business process as will be shown in the next section.

3.3 Software Architecture

The study presented in this paper is one of the case studies done as part of the DIVA project (`www.cs.vu.nl/~bastiaan/diva/`). We have been experimenting with collaborative business visualizations [4], 2D visualizations of business processes [7] and 3D VRML visualizations in a business context [6]. The goal of the project described in this paper is to find a set of 3D gadgets for the visualization of business processes. All of the case studies resulted in (prototype) implementations of visualization applications. All of them were built according the same DIVA software architecture.

The DIVA software architecture decouples information sources and visualization. This allows multiple visualizations and users to look at the same shared information source. Additionally, a single visualization can retrieve its data from multiple data sources.

The architecture distinguishes two types of data sources: static and dynamic. A typical static information source is a database. Examples of dynamic sources include simulations and measuring devices that produce new data on a timely basis. The data is transferred from its source location to the visualization components by means of the **Shared Concept Space**(SCS), a model for (distributed) data exchange.

On the client-side, a visualization application transforms the received data and updates into visual representations.

These applications should preferably be based on visualization components, such as the collection of 3D gadgets, to allow for the exchange of visualization perspectives [4].

4 Case study: Visualizing Business Processes

To experiment with the usability of the DIVA gadgets, we have created a 3D implementation of the visualizations for Gak Netherlands. The prototype consists of a 3D world which contains visualizations of the business process.

The visualization prototype is presented to the user as shown in Figure 7. The window consists of a 3D view with a few controls and a chat box. The chat box was added to enable users of the same session to chat with each other and thus aid the process of collaboration. More important is the 3D view which will contain the actual visualizations.

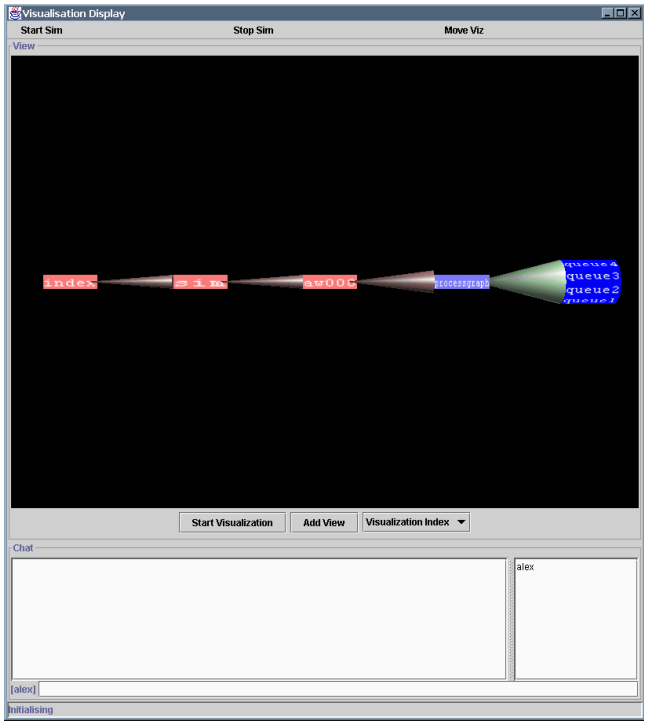

Figure 7: The initial visualization screen

Initially, the virtual world contains only a single visualization gadget, a cone tree. The cone tree visualizes the set of available visualizations. Since this prototype contains only a single simulation the set of available visualizations is limited to the process graph of that simulation and a few derived histograms. In addition to this gadget the 3D view has a few controls. First the key behavior has been used to allow navigation through the virtual world. The other controls are in the form of Swing GUI controls on the bottom of the view.

The first button *Start Visualization* is linked to the conetree indexing the available visualizations. Pressing this button results in the creation of the visualization that is currently selected in the conetree. This allows the user to start any of the available visualizations. The second button *Add View* allows users to dynamically add viewpoints to the list of available viewpoints. Pressing this button adds the current viewpoint to this list. The third control allows users to select a viewpoint from the list of available viewpoints. Once the user selects such a viewpoint the camera is moved smoothly to this new location. For each visualization gadget in the virtual world a predefined viewpoint is available. In addition to this, users can add custom viewpoints by the *Add View* button.

The most important visualization in the prototype is the process graph of the simulation. It can be created by selecting *index/sim/aw008/processgraph* in the cone tree and pressing the *Start Visualization* button. The graph gadget visualizes the flow of requests through the simulated business process. Figure 8 shows this process graph as it looks in the prototype. As we can see the general shape is about the same as the process graph in the 2D version presented in Figure 1.

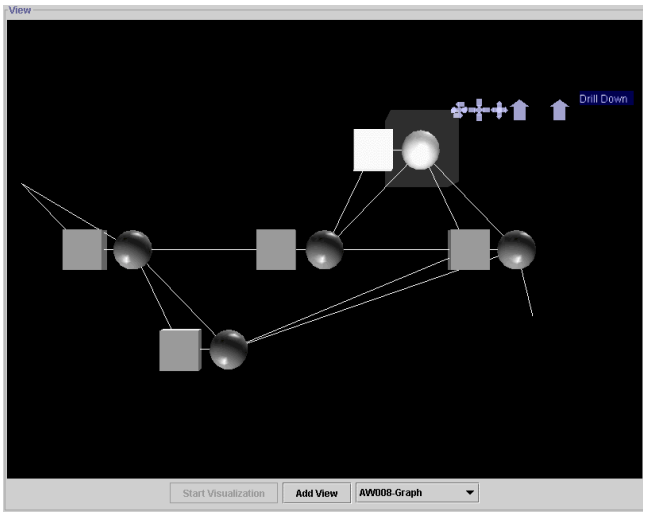

Figure 8: The process graph of which a single node is selected

For each phase in the process the visualization shows a cube and a sphere. The cube represents the queue where applications must wait to be processed while the sphere represents the actual processing. We decided to make this distinction between waiting and processing because this visualization shows the actual flow of benefit applications through the process. Benefit applications themselves are visualized as small colored spheres. When a benefit application progresses through the simulated process, the sphere that represents that application travels through the graph gadget.

The process graph makes use of the brushing behavior. When the mouse pointer is over one of the nodes (either a queue node or a processing node) or over one of the applications (or tokens, as they are called in the graph gadget) the brushing behavior displays the name of that specific element. A more interesting means of interaction is provided through the modify behavior. The graph uses the modify behavior to allow users to move, rotate and scale any of the nodes in the graph structure.

As mentioned in the discussion of the graph gadget itself the graph gadget is built up of groups. In the visualization every two nodes forming a phase are grouped together in a group named after the phase. In turn all of these phase-groups are collected in a single group, that effectively contains the whole graph. In order to manipulate these groups, the graph gadget adds a fourth and fifth button to the modify behavior. These two additional buttons respectively select the group containing the currently selected element and (de)iconify a selected group.

Another button that was added to the visualization is the *drill-down* button. This button was added to implement a drill-down feature into the visualization. Each processing node and queue node in the graph has a histogram associated with it. The drill-down button creates the histogram associated with the currently selected element of the graph. These histograms can also be reached and created by use of the cone tree index, but drilling-down on the graph provides a more natural interface to reach these histograms. Figure 9 shows the process of drilling-down on the process node for phase 4.

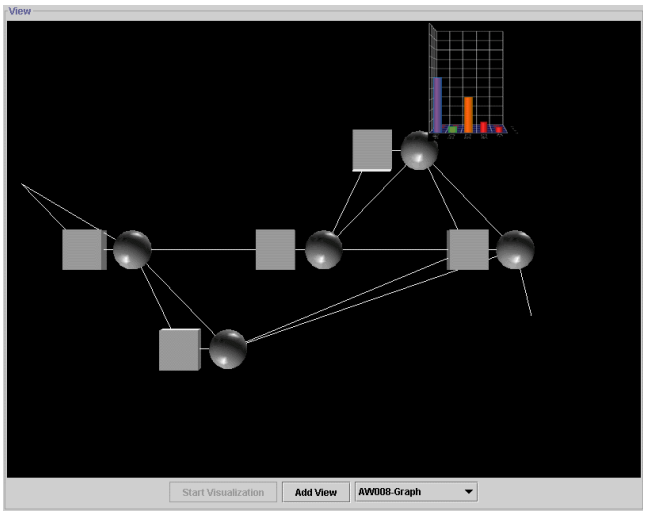

Figure 9: The drill-down button is pressed and a histogram is created

As mentioned, both queue and processing nodes of the process graph have histograms associated with them. The histograms for the queues visualize the waiting time of the tokens that have been waiting in the queue during a (simulated) week. It shows the distribution of how long applications have been in the queue during the last four weeks of simulated time. Examples of this visualization histogram can be seen in Figure 10.

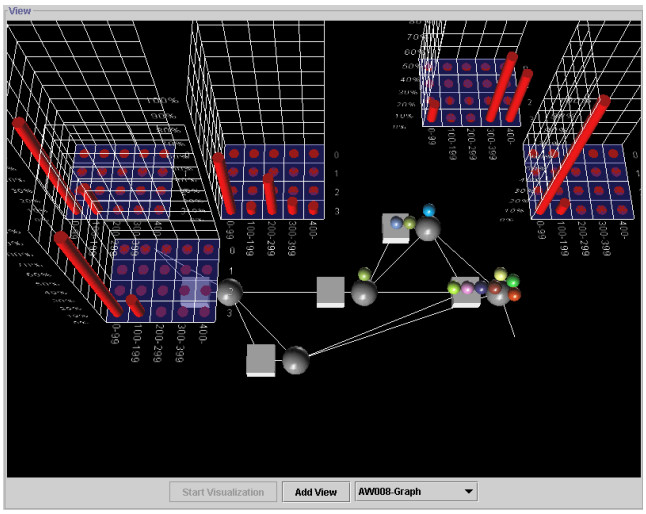

Figure 10: The process graph and associated waiting time histograms

The histograms for the processing nodes visualize the percentages of nodes that are either too early, on time or too late. These percentages are calculated by comparing the duration a certain token has been in the system to the norms for duration that have been defined for each of the phases in the process. Different colors in the bars represent whether applications are on schedule, behind schedule or late. Just as with the graph gadget, the histograms use the brushing behavior to supply additional information to the user. When a bar is brushed, the number of applications that the bar stands for is shown.

4.1 Evaluation and lessons learned

Through the application of the visualization and behavior gadgets in a real-world visualization application we were able to evaluate the flexibility and reusability of the gadgets. Additionally, we gained experience in the application of 3D techniques in a business setting.

One of the first things that we noted during the project was the fact that behavior gadgets are much more generic than the visualization gadgets. This is probably due to the fact that behavior gadgets can operate more autonomously whereas visualization gadgets largely depend on the (type of) information they will have to represent.

Another that we discovered is that there is a mismatch between simulations and (more or less accurate) visualizations in 3D space. A typical example of this problem comes from the visualization of the running simulation. In the simulation, namely, events occur when a token has moved from one stage (node) to the next. The transitions are considered as actions that take no time. The graph gadget, however, wants to visualize the transitions of the tokens by animating them as moving spheres between nodes. This movement, of course, takes time. The resulting conceptual clash could only be solved by introducing a visualization that is slightly lagging behind.

A problem we encountered during the design concerns the performance versus generality of the gadgets. Most of the time we had the choice of creating a high-performance but restricted gadget versus a slower but more general gadget. In most of the cases, the choice we made depended on the most important requirement. For example, the cone tree has to be able to contain as many nodes as possible whereas the histogram requires to be flexible with respect to how it is presented.

In comparison with the 2D predecessor of the prototype we can conclude that the 2D version is more easily accessible. The well-organized 2D visualizations in combination with the intuitive use of colors provide an uncomplicated visualization that is easily accepted by business people. The 3D visualizations on the other hand allow us to combine more information into a single scene. For example, the process graph visualizes the current status of the simulation while the histograms reveal information about the last couple of simulated weeks. In this case, the 3D visualization offers the possibilities to visualize past, present and future in a single image. What solution is to be preferred is up to the managers who have to learn to use business visualizations effectively.

5 Conclusions

In the case study performed at Gak NL, we have shown the possibility of deploying 3D visualizations in a business context. Compared to the 2D predecessor of the prototype we

have discovered both negative aspects, in terms of visual complexity and user training, and positive aspects (larger information density) of 3D business visualization.

In the case study we deployed the DIVA 3D gadgets, a reusable collection of behaviors and visualization primitives written in Java3D. They are intended as the visualization components in a software architecture that decouples information from the way it is represented in order to support networked and collaborative visualizations.

Through applying the gadgets in a business process visualization we were able to validate our requirements in a natural setting and to acquire feedback on the design-tradeoffs we encountered during the implementation of our gadgets.

References

[1] H. Koike and H. Yoshihara. Fractal approaches for visualizing huge hierarchies. In *Proceedings of the IEEE 1993 symposium in visual languages*, pages 55–60, 1993.

[2] Jakob Nielsen. 2D is better than 3D. www.zdnet.com/devhead/alertbox/981115.html.

[3] George G. Robertson, Jock D. Mackinlay, and Stuart K. Card. Cone Trees: animated 3D visualizations of hierarchical information. In *Proceedings of the ACM SIGCHI 1991 Conference on Human Factors in Computing Systems*, pages 189–194, 1991.

[4] S.P.C. Schönhage, P.P. Bakker, and A. Eliëns. So Many Users — So Many Perspectives. In *Proceedings of "Designing effective and usable multimedia systems", 9-10 September 1998, Fraunhofer Institute IAO, Stuttgart, Germany*. IFIP, 1998.

[5] S.P.C. Schönhage and A. Eliëns. Multi-user Visualization: a CORBA/Web-based approach. In *Proceedings of "Digital Convergence: the Future of the Internet and WWW", 20-23 April 1998, Bradford, United Kingdom*. British Computer Society, 1998.

[6] S.P.C. Schönhage and A. Eliëns. Dynamic and Mobile VRML gadgets. In *Proceedings of VRML99- International Conference on the Virtual Reality Modeling Language and Web3D technologies*, 1999.

[7] S.P.C. Schönhage, Ard van der Scheer, Edwin Treur, and A. Eliëns. Visualization and Simulation of Business Information at Gak NL. In *Submitted to the workshop on New Paradigms in Information Visualization and Manipulation 1999, Conference on Information and Knowledge Management*, 1999.

[8] Marc M. Sebrechts, Joanna Vasilakis, Michael S. Miller, John V. Cugini, and Sharon J. Laskowski. Visualization of Search Results: A Comparative Evaluation of Text, 2D and 3D Interfaces. In *Proceedings of 22nd ACM SIGIR conference on Research and development in information retrieval*, pages 3–10, 1999.

[9] Sunsoft. Java 3D. java.sun.com/products/java-media/3D/.