Visibility Importance Based Scene Graph Distribution Method for Distributed Virtual Environments

Ken'ichi KAKIZAKI, Kaoru SATOH
Department of Computer Science and Electronics
Kyushu Institute of Technology
680-4 Kawazu, Iizuka, Fukuoka, 820, Japan.
kakizaki@cse.kyutech.ac.jp, satoh@macross.cse.kyutech.ac.jp

Abstract. This paper describes a visibility importance based scene graph distribution method for client-server based distributed virtual environments. It is very important for distributed virtual environments that all users share the same environment on a service. In order for users to share an environment, a scene graph that describes the virtual environment must be distributed to all clients by the server. The method we propose divides a scene graph into segments and distributes these segments incrementally. The distribution is performed according to the importance of the object's visibility, and the measurement of importance is evaluated by each client. This measurement is performed in a traversing process for scene rendering, so it does not expend any extra processing cost. If an important but still as yet undistributed scene graph segment is found by a client, the client requests its distribution of the server, at which time the server distributes the segment upon demand. The method realizes an efficient scene graph distribution in a narrow bandwidth network.

Keywords: Scene Graph, VRML, Distributed Virtual Environment, Distributed Virtual Reality

1. Introduction

In recent years, implementation methods of distributed virtual environments have been well researched, and the technology is expected to become an excellent service platform in the network [Maxfield95, Broll95, Stansfield95]. Many researchers have focused on a data exchange method for dynamic information such as event data and motion data [Barrus96, Kessler96, Macedonia95, Singh95]. Efficient dynamic information exchange methods are very important: However, we believe that a scene graph distribution method is equally important for distributed virtual environments.

It is very important in distributed virtual environments that all users share the same environment on a service. In order to share the same environment, a scene graph that realizes the virtual environment must have been received from somewhere else previously. However, almost all researchers assume that the use of the distributed virtual environment will be used by the people who work together, and that the network bandwidth would be as wide as a local area network. They also assume, therefore, that a scene graph has been previously distributed to each user's machine. However, if a distributed virtual environment is to be used by an unspecified number of people, as is the case with the WWW (World Wide Web), it would be difficult to have had this scene graph distributed previously. In this case, a user must download the scene graph at the start time of a session.

Downloading is a well-known method amongst VRML (Virtual Reality Modeling Language) users, and those users also know that the download time is long, especially with a large scene graph. Users must wait until the scene graph is completely downloaded, and users cannot actually see the visual scene of the virtual environment until the download is completed. This situation can clearly be shown in the case of a narrow bandwidth connection, especially in modem based connections, and when the amount of scene graph to be distributed is large. For example, the download time for a scene graph that the amount is 10M bytes is at least one hour. This situation can lead to heavy stress for users. In order to accommodate a widely accessible distributed virtual environment like the WWW, we must introduce an efficient scene graph distribution method.

This paper describes a visibility importance based scene graph distribution method for distributed virtual environments. A scene graph for a large virtual environment is constructed from a large amount of data. However, a user can see only a small part of the environment at one time, so reference for the scene graph has locality. Therefore, the user only needs a small part of the scene graph which describes near his or her viewpoint instead of the whole of the scene graph data. We introduce a scene graph distribution method that uses the locality effectively. In order to apply the locality, our method evaluates the visible importance of objects, and distributes a scene graph incrementally, according to the object's importance.

2. Distributed Virtual Environment

In this chapter, we show the system overview we assume; we discuss a scene graph distribution method for this system.

The distributed virtual environment system is constructed based on a client-server model. The relationship between clients and servers is shown in **Figure 1**. In this system, every client can connect to all servers. This concept of connection is similar to that of the WWW. Many clients can connect to one server at a time, and the connection is used to exchange information in order to provide the distributed virtual environment service.

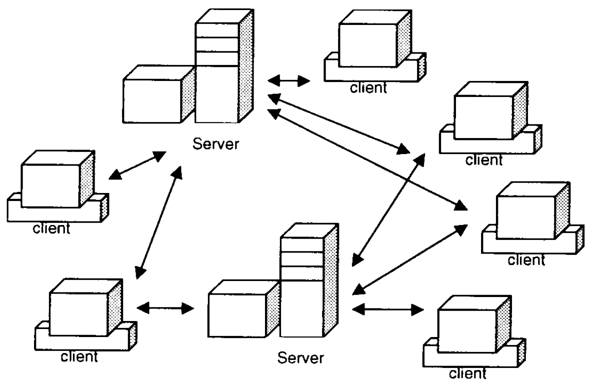

Figure 1: Client server relationship.

Each server provides an individual distributed virtual environment service to its clients. In order to provide a service, a server distributes a scene graph that describes its virtual environment, gets event information from clients, and performs simulation of the virtual environment. The system we are developing performs the simulation with a server in order to maintain the consistency of the environment.

Clients act as communication front-ends for their users. A client gets a scene graph, and renders a 3D graphics visual scene based on the scene graph. The client also handles events from input devices such as the keyboard, mouse, and joystick, and sends this information to the server.

3. Scene Graph Distribution

In this chapter, we discuss scene graph distribution methods.

3.1 Scene Graph

In this section, we review a scene graph [Hartman96, Wernecke94, Sense8-96], which we focus on in this paper.

The scene graph was originally designed for 3D graphics systems, and is now an essential form of technology for real-time 3D graphics systems, especially for virtual reality systems. The scene graph contains data about all objects in a virtual environment, such as their shape, size, color, and position. Each piece of information is stored as a node in the scene graph. In the scene graph, these nodes are arranged hierarchically in the form of a tree structure. The 3D graphics system traverses the tree structure to extract information and renders the appropriate scene based on the information.

We show an example of a scene graph in **Figure 2**. This scene graph describes a room, and the room might be a subpart of a larger scene graph. The scene graph for the room shows that there are four objects in the room. The node described "Group (Desk)" shows the visual description of the desk. This scene graph shows that the desk is constructed from two parts, one is a desk top and the other is legs. The scene graph also shows that both parts are constructed from their sub parts. The scene graph of the desk is traversed by a rendering system to render a visible scene.

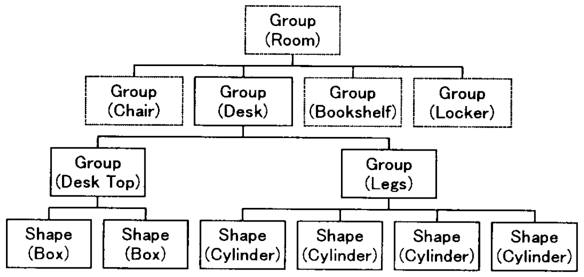

Figure 2: An example of scene graph.

In the scene graph, the Group nodes show what the parts of the objects are, so that the nodes have some sub-nodes. The Group node has information that shows:

- Position.
- Bounding box.
- Sub-node list.

The bounding box specifies an outline size of the object. The bounding box is a box that encloses the shapes in a sub-node of the node, and it is used to provide a visibility evaluation of the object to be described later.

The Shape nodes show what the visual shapes of the object is, and the nodes become terminate nodes. The Shape node has information that shows:

- Shape.
- Size.
- Color.

In the VRML[Hartman96], which is a widely used scene graph, the Group node corresponds to a Group node and a Transform node, and the Shape node corresponds to a Geometry node and an Appearance node.

3.2 Scene Graph Distribution Problem

There are some options to implementing a scene graph distribution method. In this section, we discuss two major distribution methods.

3.2.1 CD-ROM Basis

Scene graph distribution by CD-ROM is appropriate for large virtual environments, because the capacity of a CD-ROM is quite large. Some service providers of distributed virtual environments choose this distribution method for their service. The advantage of this method is that there is no need to distribute a scene graph at each connection time.

However, this distribution method has drawbacks. First, the person who wants to present a service using a distributed virtual environment must publish the CD-ROMs which contain the scene graph of his or her virtual environment. This restriction makes it impossible for ordinary people to publish their own virtual environment material on the network. Second, a person must publish a new CD-ROM when he or she changes his or her virtual environment. In addition, users may not wish to have to keep several CD-ROMs for individual distributed virtual environment services.

3.2.2 Network Basis

Scene graph distribution by network is well known as the method for VRML. This method has advantages in that anyone can access any VRML scene graph provided on a WWW server, and anyone can get the latest version of the scene graph. However, this method also has a disadvantage in that during a scene graph distribution, the user has to wait for the completion of a distribution, and usually cannot see the visual scene of the scene graph until this is done. For example, the amounts of the scene graph used by some service providers of distributed virtual environments are from 1M bytes to 20M bytes. These amounts indicate the compressed sizes of the scene graphs. The download time for an 1M byte scene graph is at least 6 minutes and the download time for 20M bytes is at least 2 hours, in the case of a user useing a 28.8K bps modem.

In order to reduce the waiting time, a method called Inline is introduced in VRML scene graph. The Inline method provides an ability to divide a scene graph into a main scene graph and segments. In this method, a user can download the scene graph, segment by segment, and a visual scene is immediately displayed on screen after the main scene graph is downloaded. Therefore, a user's waiting time is reduced if the size of the main scene graph is maintained small.

The Inline method, however, has several disadvantages. The order of the downloading of segments, for instance, is not considered, so the order is usually inefficient for a visual effect. The download completion time also becomes lengthened because each downloading of segments expends individual connection setup time. In addition, extra work is required by the author of a scene graph to divide the scene graph into its parts effectively. Moreover, the VRML has a facility to share a sub-scene graph with some scenes. However, the Inline method disables the facility between divided segments, so the total amount of the scene graph usually becomes lager.

3.3 Requirements for Scene Graph Distribution

First of all, we believe that it is very important that everyone be able to publish his or her virtual environment with ease. In addition, everyone should always have access to the latest versions of virtual environments. In order to satisfy these requests, the distribution method of scene graphs must be on a network basis instead of on a CD-ROM basis. Second, from the user's point of view, the current waiting time for the downloading of a scene graph at the start of a session must be eliminated. Third, from the system's point of view, the amount of scene graph distribution in a session must be reduced in order to reduce network traffic. In addition, processing loads must be distributed to clients as a whole instead of concentrated on a server.

4. Visibility Importance Based Scene Graph Distribution

In order to satisfy the requirements discussed in section 3.3, we propose a visibility importance based scene graph distribution method, and this method has these features:

- Partial scene graph distribution.
- Incremental segmented scene graph distribution.
- Visibility importance order distribution.
- Client-side visibility importance evaluation.

4.1 Partial Scene Graph Distribution

A scene graph for a large virtual environment is constructed from a large amount of data. However, a user can see only a small part of the environment at one time, so reference for the scene graph has locality. Therefore, the user only needs a small part of the scene graph which describes near his or her viewpoint instead of the whole of the scene graph data. Our method uses this locality to reduce the amount of data that must be distributed.

In addition, a virtual environment could be used by many users at a time. This means that these users need their own individual part of the scene graph of the virtual environment. Therefore, the part of the scene graph that is required for each user must be evaluated by a system at its execution time instead of at the design time of the scene graph. This evaluation is performed based on the visibility of objects in the virtual environment, and this process is known as culling. This culling process is performed as follows.

First, a culling process calculates a viewing frustum (**Figure 3**) which shows the viewing area of a user in a virtual environment. Next, the process checks which objects are in the frustum and which are not. The objects in the frustum are visible because they are located in the viewing area of the user. The objects outside of the viewing frustum are, on the other hand, invisible, because these objects are not located in the viewing area of a user. Our method distributes only the parts of a scene graph which correspond to the visible objects for the user.

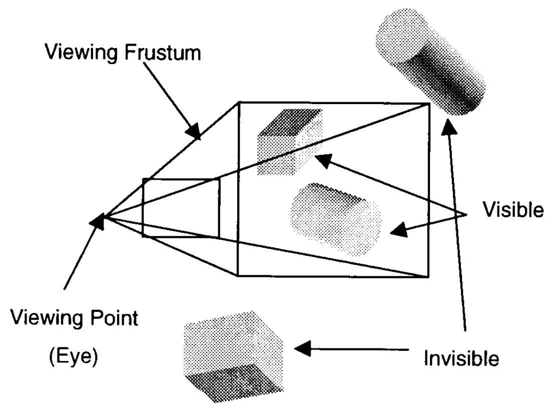

Figure 3: Visibility evaluation.

4.2 Incremental Scene Graph Distribution

In order to distribute a partial scene graph, it must be divided into a number of segments. The hierarchical structure of the scene graph is used to determine this segmentation.

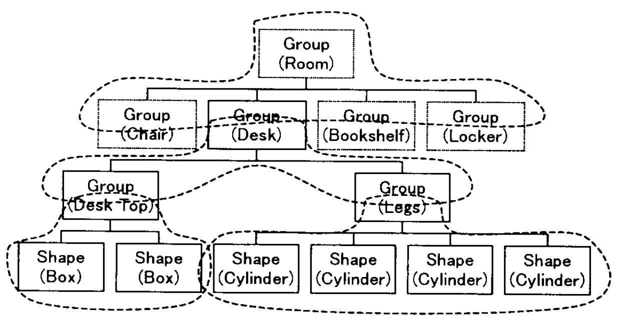

Figure 4: Scene graph partitioning.

Figure 4 shows how a scene graph is divided into several segments, and the dotted enclosures in this figure indicate divided segments of the scene graph. In this example, the scene graph is divided into four segments.

In our method, the scene graph segments that express visible objects are distributed incrementally from a server, but scene graph segments that express invisible objects are not distributed. In order to realize such selective distribution, an evaluation of the importance of an object's visibility is very important.

4.3 Evaluation of Visibility Importance

According to a visibility evaluation, there are objects chosen which should be distributed to a client. However, all chosen objects might not be distributed at one time, because the network bandwidth is limited. Therefore, we must consider the distribution order to achieve distribution effectively. In our method, the objects are distributed according to their visible importance and their distribution efficiency order, is shown below:

- The object that is located near a user in the virtual environment is important
- The object that has a large visible size is important
- The object that is described by a small amount of data is efficient for distribution

The distance of an object can be calculated from the object's position. The visible size of an object can be calculated from the object's distance and bounding box size. The distribution efficiency of an object can be calculated from the object's visible size and data size of immediate sub-nodes of the node.

4.4 Server Side Visibility Evaluation

The visibility evaluation process usually needs information regarding what objects are present and where they are located. This information is described by a scene graph. The whole of the scene graph is managed by a server, so it is quite natural that the visibility evaluation is performed by the server. However, this method has one serious handicap.

This visibility evaluation processing requires extra processing expenditure of the server because the standard processing of the server does not include such evaluation. In addition, visibility evaluation processing is required for each client that is currently connected to the server. This means that the cost of the evaluation process becomes extremely expensive when many clients are connected to the server at a time. This extra processing uses almost all the processing power of the server, causing the inability of the server to perform the simulation of the virtual environment.

4.5 Client Side Visibility Evaluation

The culling process is a basic process of scene rendering in which each client performs culling in each rendering process. Clients can, therefore, perform a visibility evaluation without any extra processing cost. In our method, we use the information produced by the rendering system to perform a visibility evaluation.

This method has strong advantages in that a processing load can be distributed to each client, and there is no extra load for the server and clients. However, this method displays a paradox. The paradox is as follows: The visibility evaluation is performed to choose what scene graph segments should be distributed. This means that the information of the chosen segments is not placed with the client, but the evaluation must be performed by the client using the segment's information.

In order to break the paradox, our method divides node information into two categories, one has a sub-node list, the other has position, bounding box, and color. The server distributes a segment of a scene graph in a form illustrated in **Figure 5**. The segment is constructed from a parent node and its immediate sub-nodes. A parent node has information about the sub-node list, and the sub-nodes have their own position, bounding box, and color. This segment form enables the culling system to evaluate the visibility of a node, and the rendering system to render a temporary shape to a screen.

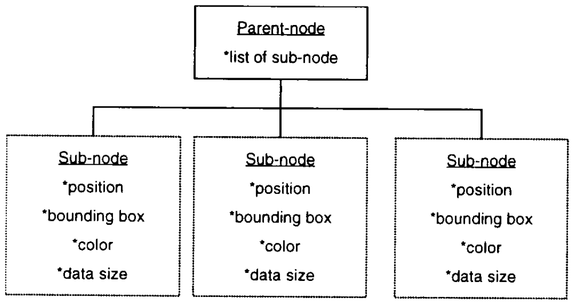

Figure 5: Distribution unit of a scene graph.

All of the scene graph segments are distributed according to the requests of a client. However, a visibility evaluation process requires a root node to start the process. Therefore, only the root part of a scene graph is actively distributed by a server except when a client connects to that server.

5. Implementation

We have been developing an evaluation system, and the system is written in Java and C.

5.1 Processing Timing

In distributed virtual environments, the service is provided on as a real-time basis. Therefore, both the simulation process in a server and the rendering processes in clients are executed as real-time processing. In order to display a visual scene smoothly, both processes are performed cyclically and continuously more than ten times in a second. This is to say that both processes are performed repeatedly in a period, and that the period is shorter than 100ms. We call this period a "Tick".

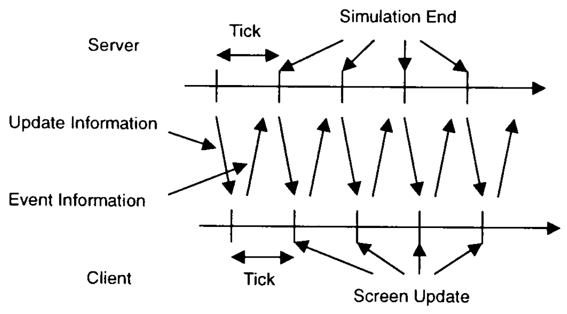

Figure 6: Processing timing.

In a server, a simulation process is cyclically performed at every Tick. In this process, the effect of event information provided by the clients is also considered. After every simulation process, information that has changed in a virtual environment such as position, and orientation of objects, is picked up. This information is distributed to clients as update information.

In a client, the update information provided by the server is used to reflect the state of the server to the client. After the reflection process, the client renders an updated visual scene. These processes are also performed cyclically and continuously at every Tick.

5.2 Client Side Processing

There are two major processes in a client, one is visual scene rendering, and the other is communication with a server.

The process flow of clients is shown in **Figure 7**, and this rendering process is performed continuously in every Tick. This process is invoked with data which refers to the root part of a scene graph in a client. In many cases, a client has only a subset of the scene graph which is managed by its server.

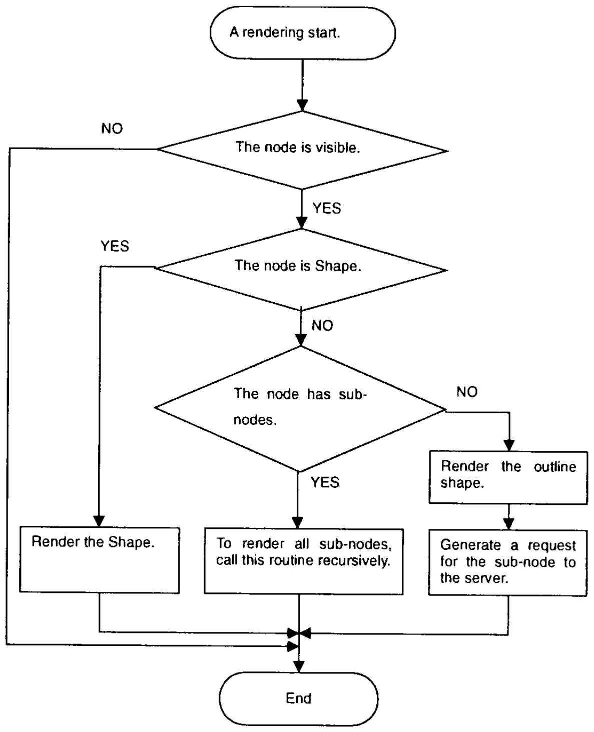
A rendering start.
The node is visible.
NO
YES
The node is Shape.
YES
NO
The node has sub-nodes.
NO
YES
Render the outline shape.
Render the Shape.
To render all sub-nodes, call this routine recursively.
Generate a request for the sub-node to the server.
End

Figure 7: Rendering process for client.

The rendering process tries to render an object in detail, but the information provided by the server would not be enough to render it in detail. The rendering process then tries to render the outline of the object based on the information that is provided by the simplified node. This simplefied node has information such as position, bounding box, and color. In addition, the client can request information from the server about a more detailed part of the scene graph of the object.

In a client, another process which communicates with a server is executed with a rendering process in parallel. This process receives the scene graph segments which are distributed by the server, and constructs a local copy of a scene graph in the client.

5.3 Server Side Processing

A server will distribute segments of a scene graph, according to a request from a client. The segments requested by the client are distributed by the server which complies with the order in the request. The distribution for each request is performed during a time that corresponds to a Tick, because new requests will come from the client at every continuous Tick. The process flow is shown in **Figure 8**.

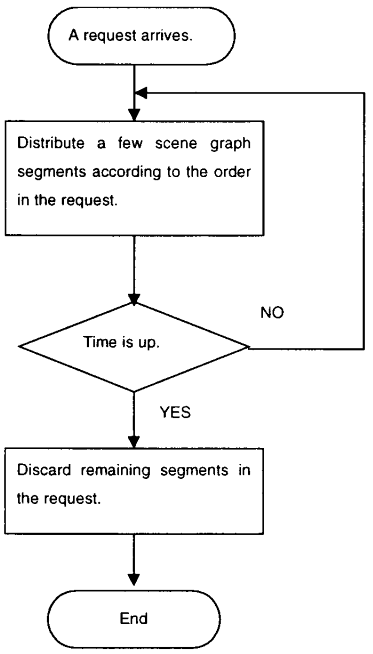
A request arrives.
Distribute a few scene graph segments according to the order in the request.
NO
Time is up.
YES
Discard remaining segments in the request.
End

Figure 8: Distribution process for server.

After the distribution, if some segments are remaining in the request, the request for segments is simply discarded. If the segments in discarded requests are important for a continuous visual scene in the client, a request for the segments can be generated by the client again. In the request, the priorities of the segments might be increased if segments that have higher priorities are already distributed according to the previous request; the request for a higher priority of segments would then be removed.

6. Example of Scene Graph Distribution

In this section, we illustrate an incremental scene graph distribution and discuss the advantages of our proposed method.

6.1 Client Startup

Initially, only the root part of a scene graph is actively distributed by a server when a client connects to that server. The root part is constructed from a complete root node and simplified immediate sub nodes of the root node. In this example, the root node is for a room shown in **Figure 9**. A dotted lined box expresses a simplified node. A solidly lined box expresses a complete node. The information stored in each node is the same as that of **Figure 5**.

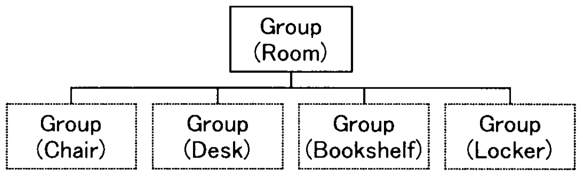

Figure 9: Initially distributed scene graph.

6.2 Rendering on the Client

According to the distributed root part of the scene graph, a rendering system of a client tries to render a visual scene. Here, we assume that only the desk is in a user's viewing area. The rendering system tries to render the desk in detail, but the information already provided by the server is not enough to render it in detail. The rendering system then tries to render the desk based on the information that is provided by the simplified node. This simplefied node has information such as position, bounding box, and color. In this situation, the desk is rendered as shown in **Figure 10**, (A). Thus, a user cannot see the detailed shape of the object, but can understand that something is there. As shown in here, the user can see an outline of an environment immediatery after connection.

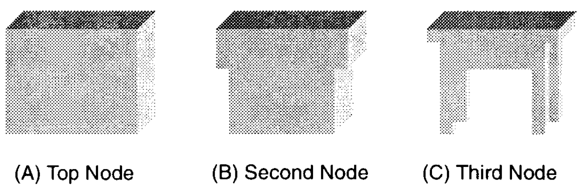

Figure 10: Incremental desk distribution.

6.3 Scene Graph Segment Request

The client can then request more detailed information about the scene graph from the server. According to the request, the server distributes the detailed part of the scene graph (**Figure 11**), the desk, to the client. The client then grafts the scene graph into an existing scene graph. **Figure 12** shows the result.

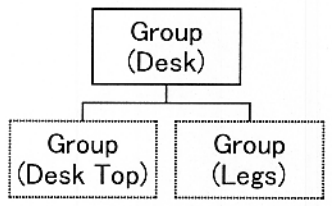

Figure 11: Scene graph segment.

As shown above, our method divides a scene graph into these segments automatically, and distributes the segments incrementally according to requests from clients. In this manner, scene graphs in clients are developed continuously, and the appearances of objects that are seen by users are developed as time goes on.

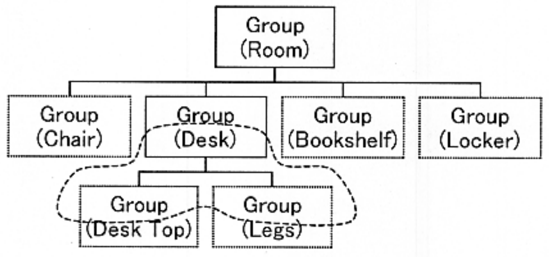

Figure 12: Scene graph grafting.

6.4 Processing in Next Cycle

In the next rendering cycle in the client, the rendering system uses the grafted scene graph shown in **Figure 12**, so that the desk is rendered from the scene graph shown in **Figure 10**, (B). Now, the user can guess that the object may be a desk or a table. The client then requests information about a more detailed part of the scene graph of the desk from the server. As a result, the scene graph shown in **Figure 13** is constructed.

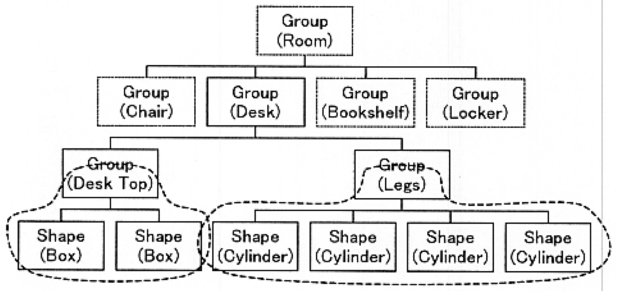

Figure 13: Completed description of a desk.

In the next rendering cycle on the client, the rendering system uses the scene graph shown in **Figure 13**, so that the desk is finally rendered completely from the scene graph shown in **Figure 10**, (C). Now, the user cleary understands that the object is a desk.

As shown in **Figure 13**, our method distributes only the required parts of a scene graph. This means that our method never distributes unnessesary parts of the scene graph. This method realizes an efficient network bandwidth utilization. In addition, the distribution of nessesary parts of the scene graph could be accelarated, because the distribution of unnessesary parts of the scene graph are automatically avoided.

Figure 14 shows another example of an incremental scene graph distribution and rendering. This example shows the distribution of a house.

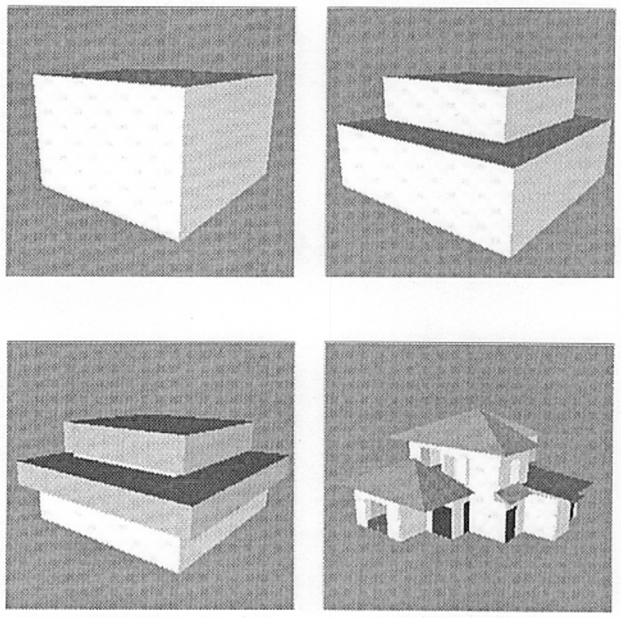

Figure 14: Example of house shape distribution.

7. Conclusion

In this paper, we proposed a visibility importance based scene graph distribution method for distributed virtual environments. The method realizes an efficient scene graph distribution in narrow bandwidth network.

References

[Maxfield95] Maxfield, J., Ferrence, T., Dew, P.: "A Distributed Virtual Environment for Concurrent Enginnering", In Proceedings of VRAIS '95, IEEE, (1995).

[Broll95] Broll, W.: "*Interacting in Distributed Collaborative Virtual Environments*", In *Proceedings of VRAIS '95*, IEEE, (1995).

[Stansfield95] Stansfield, S., Shawver, D.: "*An Application of Shared Virtual Reality to Situational Training*", In *Proceedings of VRAIS '95*, IEEE, (1995).

[Barrus96] Barrus, J., W., Waters, R., C., Anderson, D., B.: "*Locales and Beacons: Efficient and Precise Support For Large Multi-User Virtual Environments*", In *Proceedings of VRAIS '96*, IEEE, (1996).

[Kessler96] Kessler, V., D., Hodges, L., F.: "*A Network Communication Protocol for Distributed Virtual Environment Systems*", In *Proceedings of VRAIS '96*, IEEE, (1996).

[Macedonia95] Macedonia, M., R., *et al.*: "*Exploiting Reality with Multicast Groups: A Network Architecture for Large-scale Virtual Environments*", In *Proceedings of VRAIS '95*, IEEE, (1995).

[Singh95] Singh, G., *et al.*: "*BrickNet: Sharing Object Behaviors on the Net*", In *Proceedings of VRAIS '95*, IEEE, (1995).

[Hartman96] Hartman, J., Wernecke, J.: "*The VRML 2.0 Handbook*", Addison Wesley, (1996).

[Wernecke94] Wernecke, J.: "*The Inventor Mentor*", Addison Wesley, (1994).

[Sense8-96] Sense8: "WorldToolKit Reference Manual, Release 6", (1996).