A Shared 4-D Workspace

Miranda Ko

Peter Cahoon

University of British Columbia

Vancouver, British Columbia

Canada

August 25, 1995

Abstract

A Shared four-dimensional workspace is a shared animation of three-dimensional databases. Since shared animated workspaces are important to many different types of users, a pilot project was undertaken to implement a shared, time varying workspace. This software permitted the give and take of control by users running the same application across an ATM fiber link running at 100 mbits/sec. The project was divided into two parts. In the first phase, animation of three-dimensional databases in stereo was implemented. In the second phase, sharing of the animation across the ATM network was added. This paper dicusses the interfaces to the program and presents outlines of implementations of the features in each of the project's two phases.

Contents

List of Figures

1 Introduction

As medical scanning moves towards cline-clip Magnetic Resonance Imaging (MRI) and Computed Tomography (CT) scans, the need for the ability to view these live has arisen. In a similar manner, the researchers in Metals and Materials and Mechanical Engineering have the need of viewing a time history of the temperature profile for the injection moulding of aliminum cans.

Shared animation used to be a time gated sequence of two-dimensional slices. However, with the 100 Mbit ATM connections on the campus at UBC and the Vancouver hospitals, it is now possible to have shared three-dimensional and four-dimensional workspaces. The recent segmentation and reconstruction software makes it possible to view three-dimensional reconstructions in a shared workspace across the ATM network. Furthermore, in the last three months, the reconstruction was extended to include variations through time. The time axis in most of these cases is based on sampling at fixed points during the recording sequence.

In view of the demands for a shared four-dimensional workspace in both the medical and engineering fields, and with the development of the technology, a collaborative pilot project was undertaken. The project was divided into two phases. In the first phase, animation of three- dimensional databases in stereo was implemented. The second phase implemented sharing of the stereo animation across the ATM network. The basis for this shared workspace is an application called SceneViewer by Silicon Graphics (SGI). This application was implemented using Inventor, a library of objects and methods used for interactive 3-D graphics written in C++.

This paper will first briefly discusses the Inventor scene database, then present the interfaces and outlines of the implementations of different features implemented at different stages of the project.

2 Discussion of the Inventor Scene Database

The inventor scene graph is a flexible visualization tool. Since the scene graph file format is a very general description for polygonal objects in ASCII, translation of other file formats into this one is a straight forward process. The scene graph viewer used for displaying these objects comes complete with navigation tools, lighting models and matierals.

A major drawback of SceneViewer was that there is no animation support provided. Consequently, the first problem addressed by John Hogg was to redefine the node kit to contain a class that incorporated a switch node. The switch node selects which of a number of subgraphs to display. The key issue was to build a selection class that took input from a timer, slider, or a value from the keyboard.

The final version had two sliders at the bottom of the panel, one to select the step size and the other to control playback speed. There was a menu added that permitted saving the sequence as raster files, getting the static geometry associated with the scene graph, setting stereo mode, building the animation, and clearing the current geometry.

The node is the basic building block used to create three-dimensional scene databases in Inventor. Each node holds a piece of information, such as a surface material, shape description, geometric transformation, light, or camera. All 3D shapes, attributes, cameras, and light sources present in a scene are represented as nodes.[1]

An ordered collection of nodes is referred to as a scene graph. The scene graph is stored in the Inventor database. Inventor takes care of storing and managing the scene graph in the database. The database can contain more than one scene graph.[1]

3 First Phase of the Project

The first phase of the project consisted of the implementation of animation and stereo viewing.

3.1 Animation

3.1.1 Interface

The viewer provides a menu option to turn animation on and off (Figures 1 and 2).

There are scroll bars and type-in boxes for the animation step number and the speed of animation. One can go to any animation step and control the speed of animation by either dragging the scroll bar or entering a valid number directly into the type-in box. Whenever one of the scroll bar values or the type-in box value changes, the other will be updated automatically.

3.1.2 Outline of the Implementation

This section discusses how to extend the SceneViewer application to make it capable of animating. The capability of animation is achieved by modifications to the format of the data files and the SceneViewer class, which were used in the SceneViewer application.

1. **Modifications of the format of data files**

   The data files are basically the same as those used for running the SGI's SceneViewer application except that a user-defined node-kit is added to the end of the file. This node-kit is called *SeqKit*.

   Node-kits are groups of nodes. They organize a number of Inventor nodes into subgraph that has a higher level of meaning for programmers. *SeqKit* is composed of the following parts (optional - not necessary for animation):

- *seqSeparator*: It contains a SoSeparator node.

  ```
  - SoSeparator - group node that saves and restores
                  traversal state. It performs a push
                  (save) of the traversal state before
                  traversing its children and a pop
                  (restore) after traversing them. This
                  isolates the separator's children from
                  the rest of the scene graph. [2]
  ```

  seqSeparator is used to separate the animation object from the rest of the scene graph so that its own transformation, material, etc. will not affect the other part of the scene graph.

- *transform (optional)*: It contains a SoTransform node.

  ```
  - SoTransform - node defines a geometric 3D transformation
                  consisting of (in order) a (possibly) non-
                  uniform scale about an arbitrary point, a
                  rotation about an arbitrary point and
                  axis, and a translation. [2]
  ```

  transform is used to specify the transformation for the animation object.

- *seqMaterial (optional)*: It contains a SoMaterial node.

  ```
  - SoMaterial - node defines the current surface material
                 properties for all subsequent shapes. It sets
                 several components of the current material
                 during traversal. [2]
  ```

  seqMaterial is used to specify the material used for the animation object.

- *seqSelector*: It contains a SoSwitch node.

- SoSwitch - group node usually traverses only one or none of its children. There is a whichChild field that specifies the index of the child to traverse, where the first child has index 0. [2]

seqSelector is used to group all the animation steps together with the steps being the children. The whichChild field is usually initialized as 0.

- *seqGeometry (optional)*: It contains a SoSeparator node.

  - SoSeparator - see description for seqSeparator.

  seqGeometry defines the geometry for the grid of the animation object.

2. **Modifications of the class SoSceneViewer**

Several functions and an animation sensor were added to the SceneViewer class to make the SceneViewer application capable of animating when it is provided with the appropriate data files. The animation sensor is an Inventor object called SoAlarmSensor. Here is a description of SoAlarmSensor:

- SoAlarmSensor - is like an alarm clock. It is scheduled to go off at a specified time. It has a callback function associated with it. When the specified time is reached or passed, the callback function is invoked. [2]

In this program, the callback function associated with the sensor is a function that calls another function which is responsible for finding the next animation step in the data file and rescheduling the sensor.

When the user switches on animation through the interface, the callback function associated with that menu option will schedule the animation sensor to occur after an arbitrary time from now. When the time is reached, the callback function is invoked and the next animation step will be found. Then the sensor will be rescheduled.

The PSEUDO code for the function that is responsible for finding the next animation step and rescheduling the sensor is:

```
Get the current animation step from scroll bar for
    animation step number
Increment the current step
IF incremented current step exceeds maximum number of
   animation steps
     wrap around to 0
Search for SeqKit in the data file
Get part seqSelector
Set the child to be traversed to the incremented step
   number
Update scroll bar and type-in box for animation step
   number
Render screen
Reschedule sensor
```

To switch off animation, the sensor is unscheduled.

To support going to any animation step entered from the type-in box and scroll bar, a function which is responsible for finding and displaying the appropriate step in the data file was needed. It will be called whenever the type-in box or scroll bar value changes.

The PSEUDO code for the function is:

```
Get the current animation step from scroll bar
    for animation step number
Search for SeqKit in data file
Get part seqSelector
Set the child to be traversed to the value got
    from scroll bar
Render screen
```

3.2 Stereo Viewing

3.2.1 Interface

The viewer provides a menu option to turn stereo viewing on and off (Figures 1, 2, 3 top).

```
Note: The monitor must be in stereo mode while running the program.
      Two set and unset utilities were written to change the modes
      before and after viewing.
```

3.2.2 Outline of the implementation

The render area of the viewer is actually that of an Inventor viewer object. Stereo viewing is switched on and off by setting the Inventor viewer in the corresponding mode. This is done by calling a member function of the Inventor viewer.

3.3 Second phase of the project

Second phase of the project consisted of the implementations of shared animation and viewing.

```
Note: The sharing of viewing is implemented only for the
            Examiner Viewer, which is the default viewer.
```

3.3.1 Interface

Only one of the two sides in a connection can have control at any time. In particular, only one side is able to start and stop the animation and change

the direction of viewing at anytime. The window of the control side has a menu bar on top, manipulation buttons on the right, rotation sliders on the left, scroll bars and type-in boxes at the bottom of the viewer (Figure 1, top). The server window only has the scroll bars and type-in boxes at the bottom (Figure 1, bottom). When the side that has control releases control using the `Release Control` option in the topbar `Control` menu (Figure 3), the two windows will be switched.

Sharing of animation includes:
1) Starting animations on both sides at the same time.
2) When control side changes either the step number or speed by the scroll bar or type-in box, the corresponding thing will be changed on the other side.

Sharing of viewing includes:
1) When control side changes the direction of viewing, the other side will also be changed.

3.3.2 Outline of implementation

1. **Idea of implementing "Sharing"**

   Whenever something happenes on the control side, corresponding data will be written to the socket. The other side which does not have control will be checking for the presence of data in the socket continuously. The program will read and process the data if there are any.

2. **Additional functions**

   To make the program capable of sharing things between two sides, the following features were added:

   - A file that contains functions which deal with socket connections. These functions include:

(a) A function that creates a socket and then listens for connection requests.

(b) A function that connects to a specified host.

- A file that contains a function which processes data whenever the socket has data in it. Based on the data in the socket, the function determines the operations performed on the other side. Then it requests the local viewer to do the same thing by calling the appropriate function.

- An application event handler function was added to the file that contains the function "main". Since only one side is able to control the viewing of a scene at anytime, control over the viewing using the left and middle mouse buttons on the side that does not have control is disbaled. Thus, a function which processes X events differently than Inventor does is required. This is the application event handler.

  The PSEUDO code for the application event handler is:

```
IF the viewer does not have control
   IF there is a mouse button press
      IF it is left or middle mouse button press
         RETURN TRUE
         (that is the event is handled)
      ELSE
         RETURN FALSE
         (that is the event is not handled.
          Inventor will handle it afterwards)
```

- A function which updates the type-in box and scroll bar values. If the function which processes socket data determines that the corresponding values have changed on the other side based on the socket data, it will call this function with the new values.

- A function which updates the position and orientation of the camera used by the viewer. The camera is the thing that controls the direction of viewing. If the function which processes socket data determines that the camera's position or orientation has changed on the other side based on the socket data, it will call this function with the new values.

3. **Modifications of the function "main"**

Socket connection is handled by function "main".

`Note: server has control at the beginning of the program.`

The PSEUDO code to set up the socket connection is:

```
Check if program is run as server or client
IF program is run as server
        Create two sockets for listening for connection
          requests
        One of the sockets is for reading and the other is
          for writing
        Initialize viewer to have control
ELSE (program is run as client)
        Create two sockets for connections to server. The
        address of the socket for reading is the same as
        that of the server's writing socket and the address
        of the socket for writing is the same as that of the
        server's reading socket. Initialize viewer not to
        have control.
```

In addition, an application handler (refer to section **Additional Functions**) must be registered with the viewer after creating the viewer so that X events are handled properly.

Lastly, before entering Inventor's main loop as usual at the end of the function, registration of a new source of events with the Intrinsics read routine is necessary. This new source of events is the socket for reading. After the registration, whenever there is data in the specified socket, the function that processes data in socket (refer to section **Additional Functions**) will be invoked. As a result, things can be shared.

4. **Modifications of the class SoSceneViewer**

To implement the sharing of animation, the callback function associated with the menu option that turns animation on and off was modified.

The PSEUDO code for the function is:

```
IF local viewer has control
   Write data to the socket to inform the other side
     to turn on or off animation
Schedule the animation sensor if to turn on
   animation
Unschedule the animation sensor if to turn off
   animation
```

As the other side is continuously reading the socket and the time for writing to and reading from the socket is negligible, both sides start animation at approximately the same time.

To share the type-in box and scroll bar values for both the animation step number and the speed of animation, the control side informs the other side of any value changes by writing the new values to the socket. Then when the process socket data function (refer to section Additional Functions) processes that piece of data on the other side, the function that updates the scroll bar and type-in box values (refer

to section **Additional Functions**) will be called.

The sharing of viewing is achieved by attaching field sensors to the position and orientation fields of the camera of the viewer that is currently having control. These sensors are Inventor objects called SoFieldSensor. The following is a description of SoFieldSensor:

```
- SoFieldSensor - sensor class that can be attached to
                  Inventor fields. Field sensors detect
                  changes to fields, calling a callback
                  function whenever the field changes. [2]
```

The callback function in this program writes both the new position and orientation of the camera to the socket to inform the other side. Then the function that updates the camera's position and orientation (refer to section **Additional Functions**) will be called when the process socket data function processes that piece of data.

4 Future Work

Animation and viewing are the two of the features that shared across the ATM now. There are many other features that can possibly be shared. For instance, file operations (open, save, etc.), light creations and manipulations.

5 Conclusions

In this report, the methods for creating a four-dimensional shared workspace were developed. First, the interfaces and the outlines of the implementation of the animation and stereo viewing features were presented. Then the interface of the program after establishing the socket was illustrated. Finally, all the necessary changes to the program and an outline of the implementation of the sharing of animation and viewing were discussed.

6 Acknowledgements

This research was supported by the Natural Sciences and Engineering Research Council of Canada. Many thanks to John Hogg for his initial development of the Scenegraph animation software. This portion of the project was supported by Dr. Steven Cockroft in Metals and Materials Engineering and by BHP Research in Australia. Thanks to Roger Tam for his advice and help in the second phase of this project. Thanks to Stan Jang for his socket code which served as a basis for establishing the socket in this project. Lastly, I am grateful to Dr. Peter Cahoon. His suggestions and comments led to many improvements in this project.

7 References

[1] The Inventor Mentor: Programming Object-Oriented 3D Graphics with Open InventorTM, Release 2: Addison-Wesley Publishing Company.

[2] Open Inventor C++ Reference Manual.

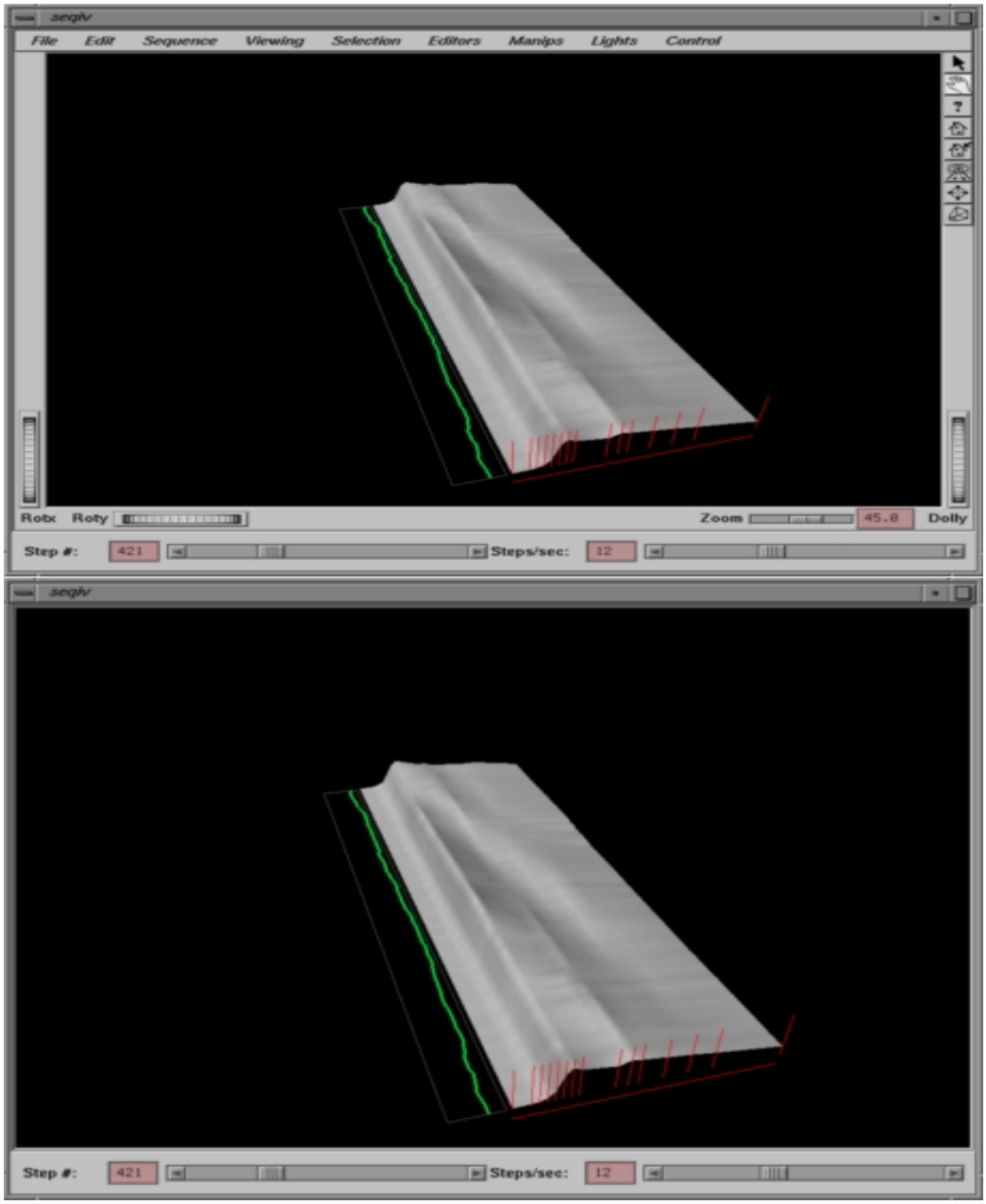

Figure 1: Top figure is the server, the bottom is the client

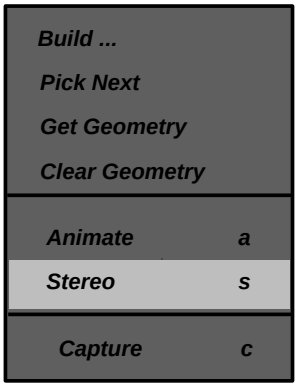

Figure 2: Mode selection

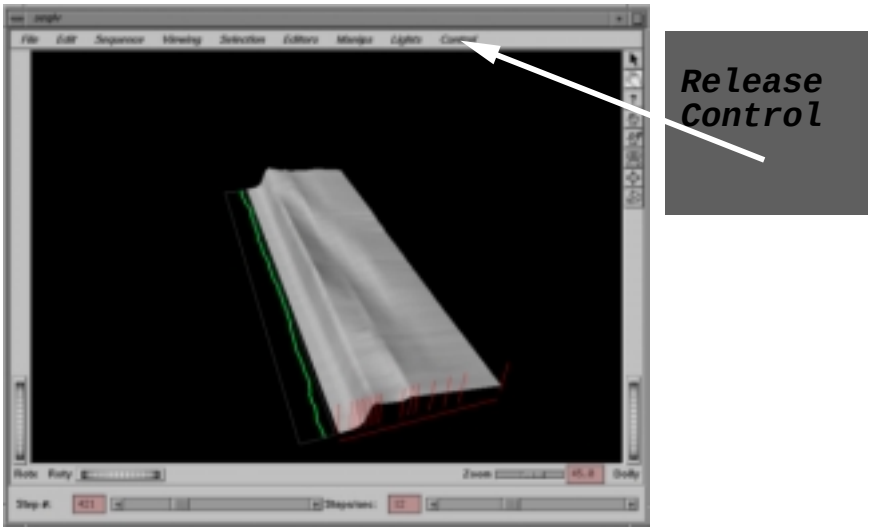

Figure 3: Control sharing