

3-D Animation for a Segmented Space Reflector Telescope

Yulu CHEN, Salvador FALLORINA, Khosrow RAD, Jane DONG, Charles W. LIU, Helen BOUSSALIS
Electrical and Computer Engineering
California State University Los Angeles, Los Angeles, CA 90032

Charalambos POULLIS
Computer Science
University of Southern California, Los Angeles, CA 90089

ABSTRACT

This paper presents the design and implementation of a 3-D animation for a large space segmented telescope test-bed. The test-bed was developed to provide a platform for experiments related to the control of precision segmented reflectors. In this paper, the physical model of the segmented telescope was imported into our 3-D animation software architecture. Algorithms based on computational geometry were developed to animate the movements of the panel segments of the telescope. In addition, a modulization of the animation system was proposed to facilitate the development and the maintenance of the animation system. This 3-D animation software architecture provides a visualization of control implementation for the segmented telescope test-bed.

Keywords : segmented telescope, 3-D animation

1. INTRODUCTION

For future space-borne astronomical missions, a segmented space reflector telescope is preferred rather than a monolithic one. To mimic the optical properties of a monolithic telescope using a segmented one, an effective real-time control system has to be established for shape control and precision pointing. Funded by NASA. A segmented reflector test-bed has been built at the Structures Pointing And Control Engineering (SPACE) Laboratory at California State University, Los Angeles, based on which, several efficient algorithms have been developed to address the problems associated with the real-time control of a large segmented optical system.

In this paper, a 3-D animation system for the segmented space telescope was developed. Such a system was used to demonstrate the effect of the proposed control algorithm and to disseminate the knowledge obtained from our research activities; this system provides the audience with different professional background with the information of the Next Generation Space Telescope (NGST.)

In this paper, we first modeled the physical system. Such a system can be controlled by a number of different control algorithms, such as PID (Proportional Integral Derivative) [1], H-infinity [2], among others. By actively controlling the position and attitude of the six peripheral segments with reference to the stationary central segment, the actuators maintain the primary mirror shape close to the perfect parabolic surface.

The physical model of the segmented telescope was imported into our software architecture. Algorithms based on computational geometry were developed to animate the movements of the panel segments of the telescope. In addition, a user-friendly graphic interface was developed for the users to specify the parameters related to the system as well as the control algorithms in an interactive fashion.

The 3-D animation software was implemented using OpenGL and Visual C++ in Windows platform. To facilitate the development and the maintenance of the animation system, we used a modulized design methodology so the animation system can be divided into four operational components – the viewer module, the 3-D-model module, the control interface module and the control algorithm module.

The rest of the paper is organized as follows: Chapter 2 introduces the physical model and the key components of the test-bed. Chapter 3 details the animation technology employed in this paper. Chapter 4 describes the OpenGL 3-D animation technology. Chapter 5 presents the 3-D modeling methods of a peripheral segment of telescope. Chapter 6 indicates the modulization of this animation system. Chapter 7 shows the results we have been achieved, and Chapter 8 is conclusion.

2. PHYSICAL MODEL OF THE TELESCOPE TEST-BED

The Structures, Pointing and Control Engineering (SPACE) laboratory Test-bed in Figure 1 is a physical model of a segmented telescope. The test-bed was used to study in an integrated way problems associated with control of large, space-borne, segmented optical telescopes such as modeling, identification, control of multi-input multi-output (MIMO) systems, structural dynamics, control-structure interaction, and disturbance rejection. The structure of the segmented telescope test-bed consists of three main parts: a primary mirror, a secondary mirror, and a truss.

Our animation tasks focuses on the movement of the primary mirror. The primary mirror consists of seven hexagonal segments mounted on a lightweight flexible truss structure. There are six peripheral segments mounted around a fixed central segment. The central segment is locked to the isolation table. The six peripheral segments are each actively controlled in three degrees of freedom (DOF's), namely piston, tilt (pitch), and tip (roll) by linear electromagnetic precision actuators. The three degrees of freedom (DOF's) are graphically illustrated in Figure 2. Each peripheral segment is attached to the truss at three node points through its three actuators. By actively

controlling the position and attitude of the six peripheral segments with reference to the stationary central segment, the actuators maintain the primary mirror shape close to the perfect parabolic surface in Figure 3.

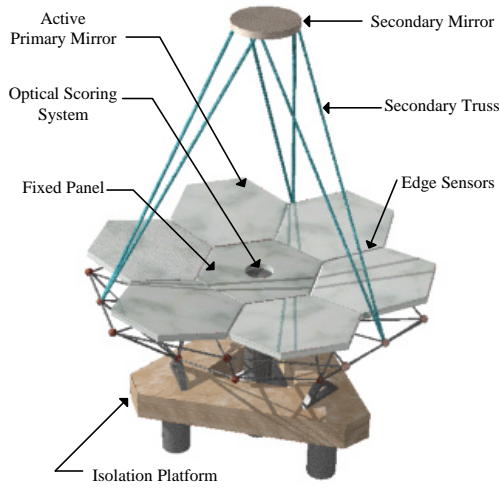


Figure 1. Features of the telescope structure

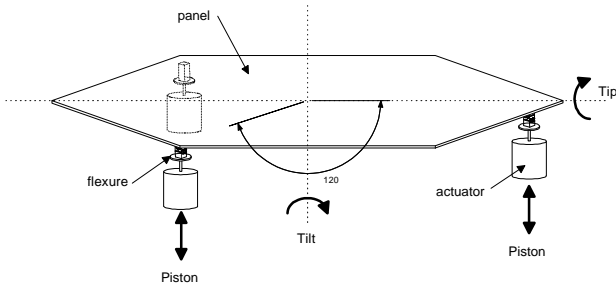


Figure 2. Panel movement in three degrees of freedom

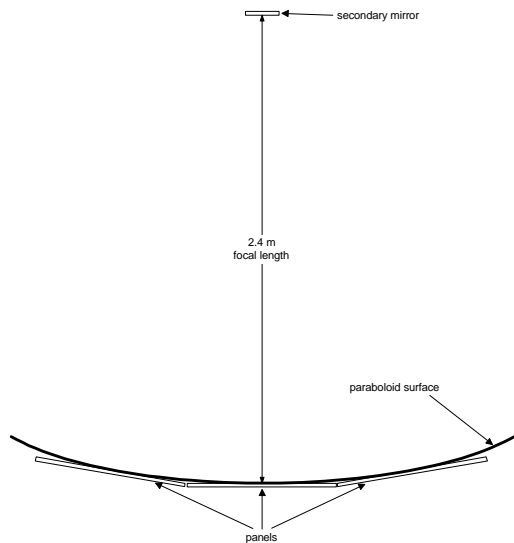


Figure 3. The primary mirrors make a perfect paraboloid surface

3. REAL-TIME CONTROL IMPLEMENTATION

Figure 4 shows the controller implementation of the physical system. The digital controller for panel shape control employs one of several control algorithms such as PID control, H-infinity control, Adaptive, and neural network control, etc. Figure 5 is a conceptual block diagram of the closed-loop system which represents the controller implementation of the physical system in Figure 4. In the block diagram, $G(s)$ is the transfer function of a physical system. $K(s)$ is the implemented control algorithm. y , y_m , d and n are the actual virtual displacements, measured virtual displacements, disturbance, and measurement noise respectively. Based on [1], the state space realization of a peripheral segment can be represented as

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} A_1 & 0 & 0 \\ 0 & A_2 & 0 \\ 0 & 0 & A_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} B_1 & 0 & 0 \\ 0 & B_2 & 0 \\ 0 & 0 & B_3 \end{bmatrix} \cdot u$$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} c_1^T & 0 & 0 \\ 0 & c_2^T & 0 \\ 0 & 0 & c_3^T \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} d_1^T \\ d_2^T \\ d_3^T \end{bmatrix} \cdot u$$

The above equations were derived from the transfer function $G(s)$ with the experimental measurements of the matrices of coefficients based on the test-bed movement.

In our 3-D animation, y provides the useful information of displacements for three actuators of the peripheral segment. We can plug the 3-D model of a peripheral segment in a block-diagram of the closed-loop system as shown in Figure 6. Our animation system can also be embedded in this closed-loop system to serve as a visualization tool. When the control algorithm is applied to the system, the closed-loop system runs the simulation of controlling movements of a peripheral segment and computes the state changes of displacement for three actuators.

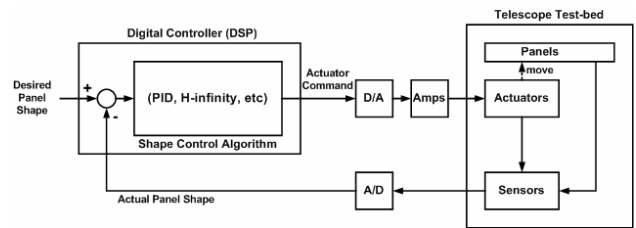


Figure 4. Real-time control implementation

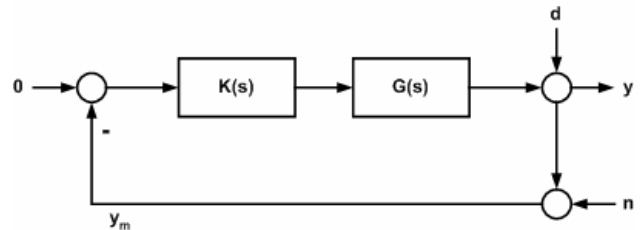


Figure 5. Block diagram of the closed-loop system

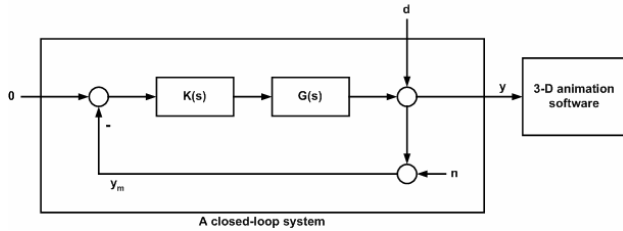


Figure 6. Embed our 3-D animation software to the closed-loop telescope test-bed system

4. ANIMATION TECHNOLOGY

The 3-D animation of telescope was implemented using OpenGL and Visual C++ in windows platform. OpenGL is a software interface to graphic hardware. It is designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms. OpenGL is not a programming language. It is a 3-D graphics and modeling library like the C language runtime library. Programmers can use OpenGL's prepackaged functions to specify the objects and operations which are needed to produce interactive three-dimensional applications.

With OpenGL, users build up desired models from a small set of geometric primitives – points, lines and convex filled polygons. In addition, OpenGL supports lighting and shading, texture mapping, blending, transparency, animation, and many other special effects and capabilities.

The base OpenGL library provides this powerful but primitive set of rendering commands only, so all higher-level drawing programmed using these commands. Numbers of libraries based on OpenGL exist to help programmers to simplify programming task. The OpenGL Utility Library (GLU) which does more complex tasks is the common useful library accompanied with OpenGL. For example, the basic OpenGL library does not support 2D or 3-D quadrics objects such as spheres, cylinders or disks which are calculated with quadric equations, but GLU supports these quadrics objects. [3][4]

5. THE 3-D ANIMATION OF SEGMENTED TELESCOPE

To implement the 3-D animation of segmented telescope, we built the 3-D model using OpenGL geometric primitives. All geometric primitives are described in terms of their vertices. We can use simple OpenGL primitives to compose complex objects. A 3-D model structure is composed of a number of polygons. The polygons represent the areas enclosed by single closed loops of line segments and drawn with pixels in the interior filled in.

A peripheral segment of primary mirror is a hexagonal panel mounted on three actuators with three I-flexures in Figure 7. We use polygons as 3-D models' surfaces to assemble these objects. For example, a hexagonal panel is assembled with two hexagonal and six rectangular polygons. An actuator is created with cylinders and disks.



Figure 7. Components of a peripheral segment

Each component of a peripheral segment has specific function, material property and motion. For example, the segment position actuators are identical linear electromagnetic force actuators which get the commanded displacements for the shape control. The actuator controls panel's movement by shifting a shaft up and down. The hexagonal panel is defined as a rigid body. The behavior of an I-flexure is like an elastic column. We also need to define some major properties of these components for kinematics model of animation.

We use OpenGL's modeling transformations to create the animated effect. Three modeling transformations are translation, rotation and scaling. These commands provide model-view matrices for transforming coordinate system. The vertices of primitives are used as a single-column matrix and multiplied by the model-view matrix to generate new transformed coordinates.

As we animated an actuator in Figure 8, we translated the shaft along z axis. When one of the three actuators is activated, it drives the movement of the hexagonal panel and bends the I-flexures. We used translation and rotation to animate the movement of the hexagonal panel. The stroke of an actuator is +/-2.5mm; that is, the maximum displacement is 5mm which compares to the dimension of hexagonal panel (each side is 500 mm) is very small. It makes the bends of the I-flexures ignorable.

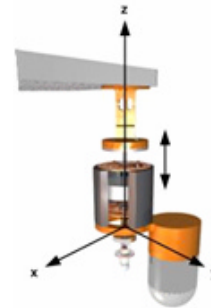


Figure 8. The movement of an actuator

The movements of the six peripheral segments are determined by the control algorithm as discussed in Section 3. In each iteration of processing, the control algorithm is performed based on the errors received from the sensors. Then, the real-time control algorithm generates the state changes on each of the peripheral segments. Based on the closed-loop system generates the computational displacements of each actuator to mimic the physical behavior of the test-bed. Such displacements are sent to our 3-D animation system as an array of input data.

In our animation, we use the array of data to re-draw the position of each of the panels for animation. If the displacements are large, interpolation is used to draw a number

of intermediate positions before the panel reaches its final position.

The control algorithm only gives the displacements of actuators but the movements of panels have to be modeled based on the kinematics property of the physical system. Since the scale of displacement is very small relative to the dimension of hexagonal panel, computational geometry model are used instead of kinematics to simplify the animation process.

The simplified animation of the movement of a peripheral segment is shown in Figure 9. Three actuators of a peripheral segment receive displacements, $d1$, $d2$, $d3$, respectively, to move the panel from plane p to plane p' . Assume that $d1 < d2 < d3$. The computational steps are itemized as follows:

Step 1: Triangle $\Delta a_0 b_0 c_0$ is shifted to Triangle $\Delta a_1 b_1 c_1$ using OpenGL translation function; with the displacement of, $d1$, where $d_1 = \overline{a_0 a_1} = \overline{b_0 b_1} = \overline{c_0 c_1}$. This step is shown in Figure 10.

Step2: Move Triangle $\Delta a_1 b_1 c_1$ to Triangle $\Delta a_1 b_2' c_1$ by rotating against $\overline{a_1 c_1}$. This rotation should follow the curve $b_1 b_2'$, so the Triangle $\Delta a_1 b_2' c_1 = \text{Triangle } \Delta a_1 b_1 c_1 = \text{Triangle } \Delta a_0 b_0 c_0 < \text{Triangle } \Delta a_1 b_2 c_1$ and on the plane of Triangle $\Delta a_1 b_2 c_1$. This step is shown in Figure 11.

Step3: Move Triangle $\Delta a_1 b_2' c_1$ to Triangle $\Delta a_1 b_2' c_2'$ by rotating against $\overline{a_1 b_2'}$. This rotation should follow the curve $c_1 c_2'$. so the Triangle $\Delta a_1 b_2' c_2' = \text{Triangle } \Delta a_0 b_0 c_0 < \text{Triangle } \Delta a_1 b_2 c_2$ and on the plane of Triangle $\Delta a_1 b_2 c_2$. This step is shown in Figure 12.

Step4: $\Delta a_1 b_2 c_2$ is the new position for a hexagonal panel. To centralized the panel after the changes of the position achieved from the first four-step, the center of the resulting plane is aligned with a fixed reference pivot of the segment as shown in Figure 12. This step is critical to avoid the deviation of the panel due to computational approximation.

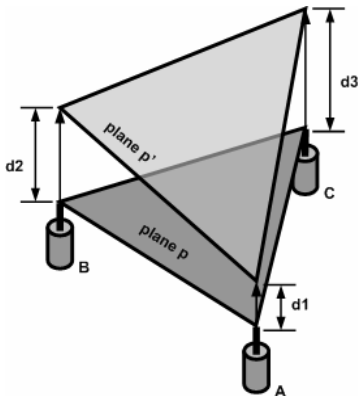


Figure 9. The geometric movement of a peripheral segment

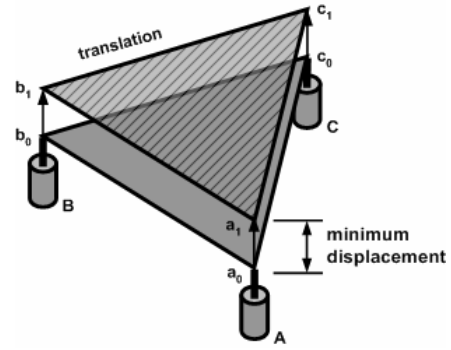


Figure 10. Implement 3-D animation – step 1

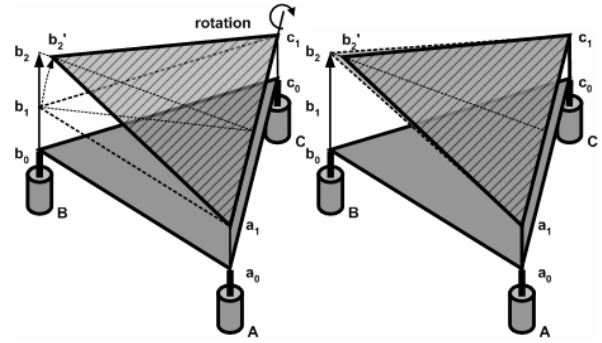


Figure 11. Implement 3-D animation – step 2

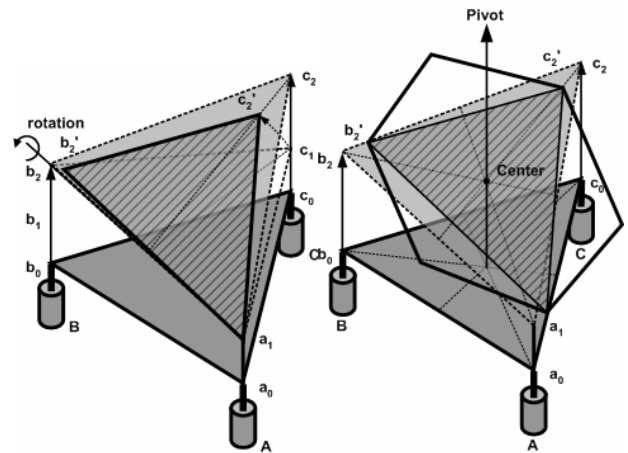


Figure 12. Implement 3-D animation – step 3, 4, 5

6. MODULIZATION OF THE ANIMATION SYSTEM

To facilitate the development and the maintenance of the animation system, we divided the system into four operational components – the viewer module, the 3-D-model module, the control interface module and the control implementation module. The organization of the system and the interactions among the modules can be seen in Figure 13. The functionalities of the modules are described below.

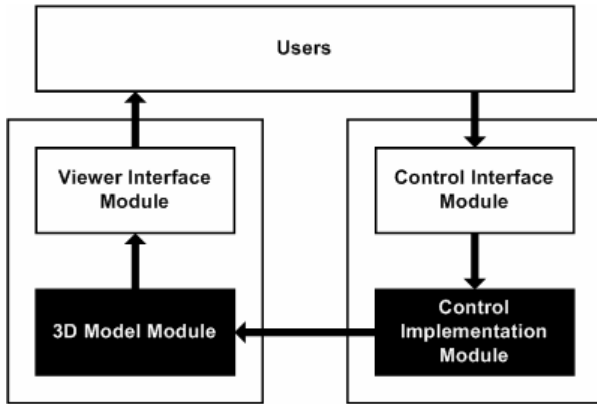


Figure 13. Modulization of the animation system

The Control Interface Module

The control interface module is a graphic user interface designed using Microsoft Foundation Classes (MFC). It allows users to setup control algorithms and application-specific input disturbances. The control interface module defines the formats of inputs, collects necessary data and parameters from the user, and then sends them to the control algorithm module. The purpose of this module is to support a well-defined and user-friendly interface. Also, the control interface module can be revised based on the requirements of different applications. The other components shielded by the interface can be kept intact to further ease the design of the system.[5][6]

The Control Implementation Module

The control implementation module does the major computation works. This module receives the data or parameters from the control interface module and computes results using a control algorithm chosen by the user. The control implementation module simulates all the control devices of the telescope test-bed and generates decentralized control in an iterative fashion. In our animation system, the output of decentralized control is converted to an array of relative displacements due to the movement of the actuators for the 3-D model animation. I.e., such information sent to the 3-D-model module is an array of vertices represented by a (x, y, z) coordinate. The vertices form the polygon shapes of the telescope panels and are employed to our 3-D animation.

The 3-D-Model Module

The OpenGL routines in the 3-D-model module receive vertices and compose 3-D models using basic geometric primitives such as points, lines and polygons. The kinematics properties of 3-D model also define in this module using OpenGL modeling transformations. For keeping this module independent from mathematical computing, we create geometric primitives and define kinematics properties only and leave input ports of transformations for computing results of control implementation module.

The Viewer Module

The viewer module is a user interface for showing the animation of 3-D telescope models. For observing the animation, the viewer module should have basic viewing functions such as zoom, rotation and pan. The graphic user interface (GUI) is developed using Microsoft Foundation Classes (MFC) and the viewing functions are developed using OpenGL; that is, it should integrate MFC and OpenGL together.[7]

The rotation and pan functions use two of OpenGL modeling transformations - rotation and translation. We create the zoom function using perspective projection. The viewing volume for a perspective projection is a frustum of a pyramid. The area of the back of the viewing volume is bigger than the area of the front. Thus, an object appears larger at the front than at the back after mapping the viewing volume to the viewport. We move an object along z axis to implement zoom in or zoom out.

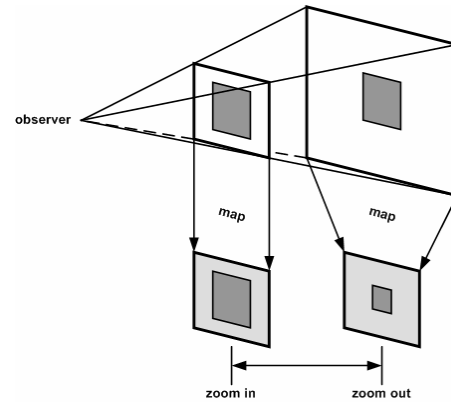


Figure 14. The implementation of zoom function

7. IMPLEMENTATION AND RESULTS

The 3-D models of an actuator, an I-flexure and a hexagonal panel are shown in Figure 15, Figure 16 and Figure 17 respectively. We use these components to build a 3-D model of a peripheral segment which is shown on viewing window. Viewing window supports real-time viewing functions. By clicking and moving a mouse, users can implement zoom, rotate and pan functions and the viewing results will be shown immediately. The results are shown in Figure 18 and Figure 19.

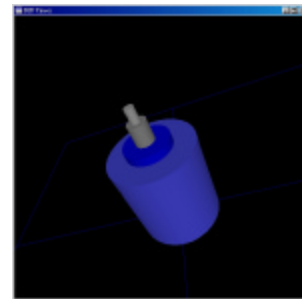


Figure 15. The 3-D model of an actuator

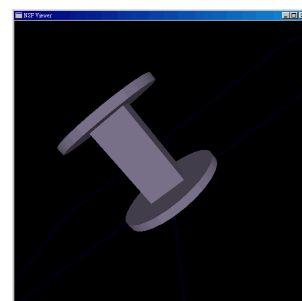


Figure 16. The 3-D model of an I-flexure

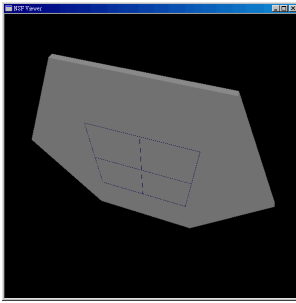


Figure 17. The 3-D model of a hexagonal panel

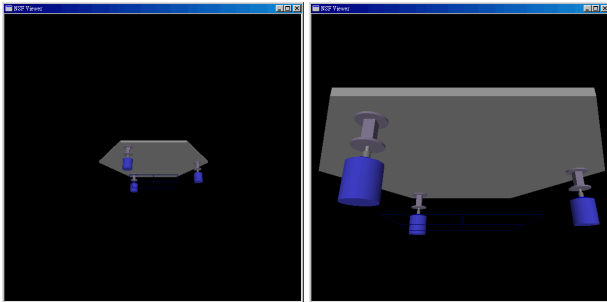


Figure 18. Zoom out and zoom in functionality

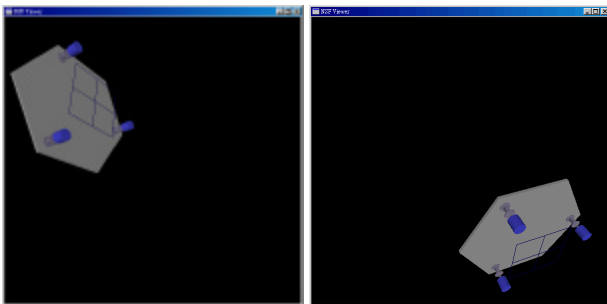


Figure 19. Rotation and pan functionality

8. CONCLUSION

This paper presents the design and implementation of a 3-D animation of a large space segmented telescope. The 3-D animation software was implemented using OpenGL and Visual C++ in windows platform. OpenGL provides powerful computational geometric primitives, modeling transformation and rendering effects to support the development of our 3-D animation. To further facilitate the development and the maintenance of the animation system, we divided the system into four operational components – the viewer module, the 3-D-model module, the control interface module and the control algorithm module. With the modulization, different control algorithms, as well as computational geometric models can be ported into their corresponding modules. Thus, the rest of the system will be intact. In the future, the 3-D animation software will serve not only as the simulation software but also will be embedded in the SPACE test-bed.

9. REFERENCE

- [1] H. Boussalis, M. Mirmirani,, Z. Wei, “Decentralization and PID Controller Design for Large-Spaceborn Telescopes”, IASTED International Conference on Application Modeling, Simulation and Optimization, Pittsburgh, 1995.
- [2] H. Boussalis, M. G. Raghavender, “The Application of H-infinity Control Law to a Decentralized Segmented Reflector”, IASTED/SCS Conf., Australia, 1996.
- [3] OpenGL Architecture Review Board , 1999, **OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.2, 3rd Edition**, Addison-Wesley Pub Co.
- [4] Richard S. Wright Jr., Michael R. Sweet, 1999, **OpenGL SuperBible, 2nd Edition**, Waite Group Pr.
- [5] Herbert Schildt, Frank Crockett, 1998, **MFC Programming from the Ground Up, 2nd Edition**, McGraw-Hill Osborne Media.
- [6] Jeff Prosise, 1999, **Programming Windows With MFC, 2nd Edition**, Microsoft Press.
- [7] Ron Fosner , 1996, **Opengl Programming for Windows 95 and Windows NT**, Addison-Wesley Pub Co.