THE GATI CLIENT/SERVER ANIMATION TOOLKIT

Chandrajit L. Bajaj
Steve Cutchin

CSD-TR-92-096
December 1992

The GATI Client/Server Animation Toolkit*

Chandrajit L. Bajaj *Steve Cutchin*

Department of Computer Science,
Purdue University,
West Lafayette, IN 47907

Tel: 317-494-6531
Fax: 317-494-0739
email: {bajaj,cutchin}@cs.purdue.edu

Abstract

This paper presents GATI, an animation server that provides for, distributed, potentially collaborative, real time interactive animation in two and three dimensions. The system supports a high level animation language based upon a commands/event paradigm. Examples are given of how the toolkit is being used in a distributed, collaborative geometrical modeling environment. GATI runs on unix platforms supporting the X-11 windowing environment and using the XS Graphics Libraries.

1 Introduction

Recently there has been great interest in algorithm animation, user interface animation and key-frame animation. Algorithm animation and user interface animation have concentrated almost exclusively on two-dimensional animation and key-frame animation systems have put great effort into generating photo-realistic three-dimensional animations that require hours per frame to generate. There exists in the area of computational geometry many problems that deal with a moderate number of graphical objects in three dimensions. With todays powerful graphics workstations it is longer necessary to limit animation to two-dimensions or to be satisfied with prolonged frame generation sequences for this particular class of problems.

*Supported in part by NSF grants CCR 90-00028, DMS 91-01424 and AFOSR contract 91-0276

We present in this paper GATI, a system for general purpose three-dimensional interactive animation that runs on various hardware platforms and provides close to real time three-dimensional animation services. GATI should be very useful in many areas of computational geometry and solid modeling. It could prove to be very useful as a visual debugger. GATI is an animation toolkit developed within the frame-work of a collaborative, distributed geometric software environment called SHASTRA[2]. The SHASTRA software environment provides a powerful substrate of tools for collaborative and distributed work, it also provides advanced networking facilities that ease the task of communicating between applications.

This paper provides an overview of the organization of the GATI animation system plus presents examples of how the system has been used to ease the creation of several geometric modeling and robotics applications and to provide animation services to other SHASTRA applications.

2 Related Work

Much recent research has been done in the area of algorithm animation and animating user interfaces. Balsa[4] is a well known algorithm animation system. Tango[14] is another algorithm animation system based upon a path-transition paradigm. Both of these systems provide excellent facilities for generating two-dimensional animations of algorithms.

With respect to animated user interfaces Whizz[5] typefies a system for developing an animated interactive application. Whizz is based on a streams/event model and is limited to two-dimensional animation.

In the area of three dimensional animation systems a host of systems are available. Powerful keyframe animation systems are available from such companies as Wavefront Technologies Inc[9]. Alias Research Inc.[6] Abel Image Research[12], Vertigo Systems Inc.[8], Symbolics Inc.[7] and others. Also a collection of animation systems have been developed at various universities as part of ongoing research projects into motion planning, dynamic simulation and animation scripting systems[13][11]. There is also DYNAMO[10] a system for generating dynamic simulations via kinematic constraints, behavior functions, and inverse dynamics. However most of these systems are single user, single workstation programs. In contrast GATI can simultaneously service multiple

clients and allow multi-user interactions. Furthermore GATI can asynchronously support multiple views and can distribute its rendering over a network of workstations for distributed parallelism.

SHASTRA[2] is a highly extensible, distributed and collaborative geometric software environment consisting of a growing set of individually powerful and interoperable (client-server) toolkits which support collaborative design sessions. In the SHASTRA environment, the application toolkits listed below, run as independent processes on separate workstations having separate user interfaces (using X-11 and Motif). The application toolkits make use of a custom designed network library to communicate data structures conveniently between each other and manage multiple connections across a network.

1. The GANITH algebraic surface modeling toolkit provides symbolic and numeric computations on algebraic varieties.

2. The SHILP solid modeling and display toolkit manipulates curved solid objects with piecewise algebraic surfaces.

3. The VAIDAK medical imaging and model reconstruction toolkit manipulates medical image volume data.

4. The BHAUTIK physical analysis toolkit provides a graphical interface and functionality to set up and perform scientific and engineering simulations on geometric models.

GANITH provides the surface modeling infrastructure for SHILP and VAIDAK. Further, SHILP provides all the solid model manipulation and display functionality for skeletal structures reconstructed from CT/MRI image data in VAIDAK. GANITH, SHILP and VAIDAK provide BHAUTIK with a varied source of geometric domains. Collectively these toolkits provide a vast modeling infrastructure. GATI interoperates with GANITH, SHILP, VAIDAK and BHAUTIK toolkit processes and acts as their animation server.

3 System Model - Commands/Events

The GATI animation paradigm is based on the basic client/server model. Animation applications are the clients of an animation server that accepts commands in a high-level animation language. The animation server generates animation events that are sent to its clients. These events can be caused by mouse events (button press, motion) in animation windows or can be caused by the interaction of various animation objects within the animation environment. For example when an object has entered a certain region of a coordinate system or if an object has completed a specific animation command. This second set of events is useful in providing applications with the ability to interact asynchronously with the animations generated. A client can begin an animation and be signaled when the animation has completed.

The server can provide animation to multiple clients. The clients can each either generate separate animations or can collaborate on a single animation. This can be useful for a set of multiple processes working on a single task or possibly for a set of distributed interactive systems that each generate a portion of the animation.

This animation server to animation clients paradigm fits in nicely with the distributed and collaborative nature of the SHASTRA environment.

4 System Overview

The GATI system consists of three primary components: An interactive animation server that is responsible for displaying the animations, A user library that is linked to animation clients to simplify communication between the server and the users client, and finally an interactive graphical editor that provides facilities for creating preset animations that can be used 'off-the-shelf' in user programs. These three components are described in detail in the following sections.

4.1 Animation Server

The GATI server provides interactive animation services to other SHASTRA applications, potentially even other GATI servers. The server accepts commands in a high level animation language

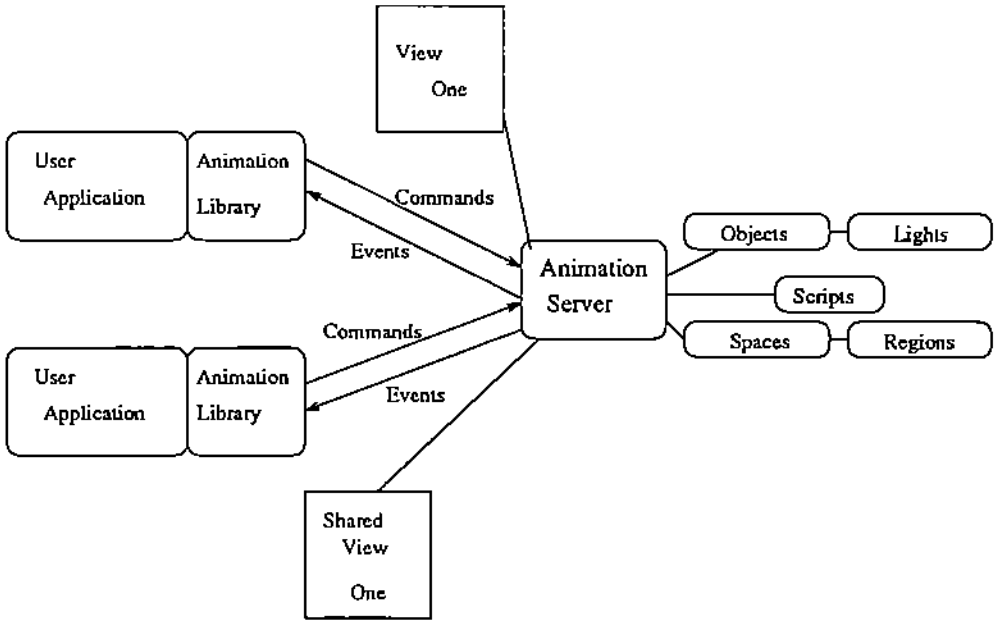

FIGURE 1: The Software Architecture of GATI

and can generate animation events in response to user input and various graphical events. It supports multiple clients that can either share animations or have there own separate animations. It provides facilities for generating frame-by-frame movie files that can be used to play-back animations quickly or to use more powerful rendering programs to generate photo-realistic frames that can be saved on video-tape for high-quality animations.

4.1.1 Animation Language

The animator accepts commands from other SHASTRA applications in a high-level textual language that can be used to define complex animations. The language allows the definition of **movies** with a set of **objects**, **lights**, **spaces**, **scripts**, **views**, and **regions**. A host of operations can be performed on each of these items. The nature of these items are defined below.

object An object is a two or three dimensional graphical object represented internal as a set of polygons. Associated with every object is a collection of attributes that can alter the appearance of that object in an animation, for example color, material, orientation, etc.

lights The animator supports two kinds of light sources. Either point lights or infinite lights. Point lights exist within a coordinate system and can be seen in an animation. As such point lights can have polygonal representations associated with them. Infinite lights have no coordinate position and are represented primarily as a direction vector. Thus it is not possible to see an infinite light source. Only the effect that the light it generates is visible in a view.

spaces A space is an organizational device that allows for the creation of hierarchical models for animation. It also provides facilities for mixing two-dimensional animations and three-dimensional animations in a single animation. A space defines a local reference frame that can contain **objects**, **lights**, **regions** and other **spaces**. Spaces can contain both two-dimensional and three-dimensional **objects** and **spaces**. This ability to mix two and three dimensional animations can provide some powerful animation effects.

scripts scripts are nothing more than user-defined animations. A collection of animations can be collected together into a single script that can be later applied to multiple objects. This simplifies animation development, and provides simple mechanisms for having a collection of items undergo identical animations.

views views are actual on-screen windows and can be used to look at animations from different positions, angles and with different projections (orthographic versus perspective).

regions are used to mark subsections of spaces for event management. If an **object** enters or exits a **region** an event is triggered and sent to the client responsible for the particular animation. Events are discussed in more detail in the following section.

4.1.2 Animation Events

The animation server provides for two types of events: graphic events and mouse input events. A graphic event occurs when an **object** enters/exits a region or when an object completes a specific animation sequence. The occurrence of a graphic event causes a message to be sent to all clients that can access the **object** that triggered the event. The second class of events, mouse events, occur when a button on the mouse is pressed within an animation view or the mouse is moved within an

animation view. A mouse event causes the space, object, position, and button information to be passed on to the clients that can access the view in which the event occurred.

Events must be turned on by sending explicit commands to the server. These animation events provide for interactive manipulation of animations. For example by using a region event simulation of course grain collision detection can be created at little cost to the applications developer.

4.1.3 Animation commands

The animation language provides commands for creating all of the previously described objects as well as performing animations on them. It can perform rotations, translations, and scaling of objects , spaces, and lights. It supports changes in color and shade of objects and lights. Scripts can be defined and applied to objects, spaces, lights, and scripts. Events can be specified for objects, , lights, and regions. Recording can be turned on for any number of views so that an animation in a particular view can be saved and played back later.

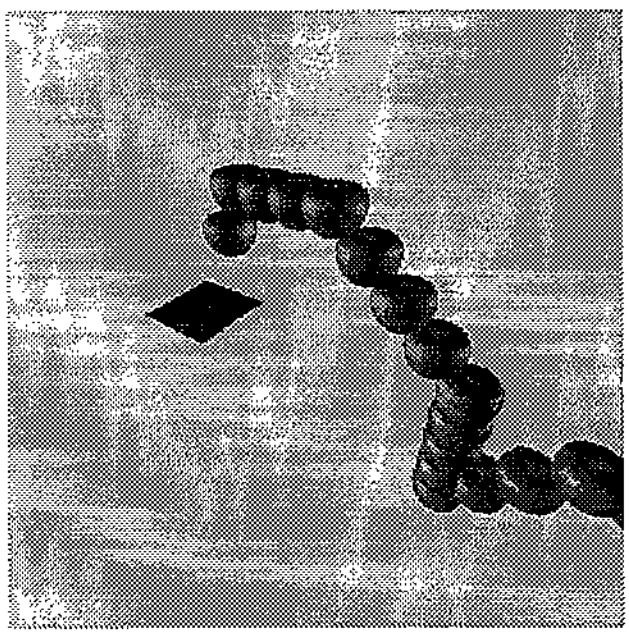

FIGURE 2: An apple moving along a simple path

```
# - a sample gati script for moving an apple

create gobject "apple" "apple.poly" ;
create gobject "square" "square.poly" ;
create view "bouncer" ;
create space "dancer" ;
create space "suber" ;
gobject set space "apple" "suber"  ;
gobject set space "square" "dancer" ;
space set space "suber" "dancer"  ;
view set space "bouncer" "dancer"  ;
translate space "dancer" ( -4.0 0.0 0.0 ) 1 ;
scale space "dancer" ( 0.5 0.5 0.5 )  1 ;
scale gobject "apple" ( 0.8 0.8 0.8 ) 2 ;
scale gobject "square" ( 2.5 2.5 2.5 ) 1 ;
translate gobject "apple" ( -1.0 1.0 0.0 ) 1 ;
define "curve" ( "obj1" )
    translate space "obj1" ( 1.6 1.6 0.0 ) 3;
    translate space "obj1" ( 1.6 0.0 0.0 ) 3;
    translate space "obj1" ( 1.6 -1.6 0.0 ) 4;
    translate space "obj1" ( 0.0 -1.6 0.0 ) 4;
 enddef

#now apply the curve to the object a few times

apply "curve" ( "suber" ) ;
apply "curve" ( "suber" ) ;
apply "curve" ( "suber" ) ;
```

FIGURE 3: A sample GATI script

4.2 User Library

The user library is a C library that must be linked to an application for it to take advantage of the Animation server. It provides utilities for sending animation commands to the server, specifying callback functions to handle animation events, loading saved objects, loading and saving animation scripts to files.

4.3 GATI Animation Editor

The GATI animation editor is an interactive graphical editor that allows animators to create and edit animation scripts that can be saved for later use.

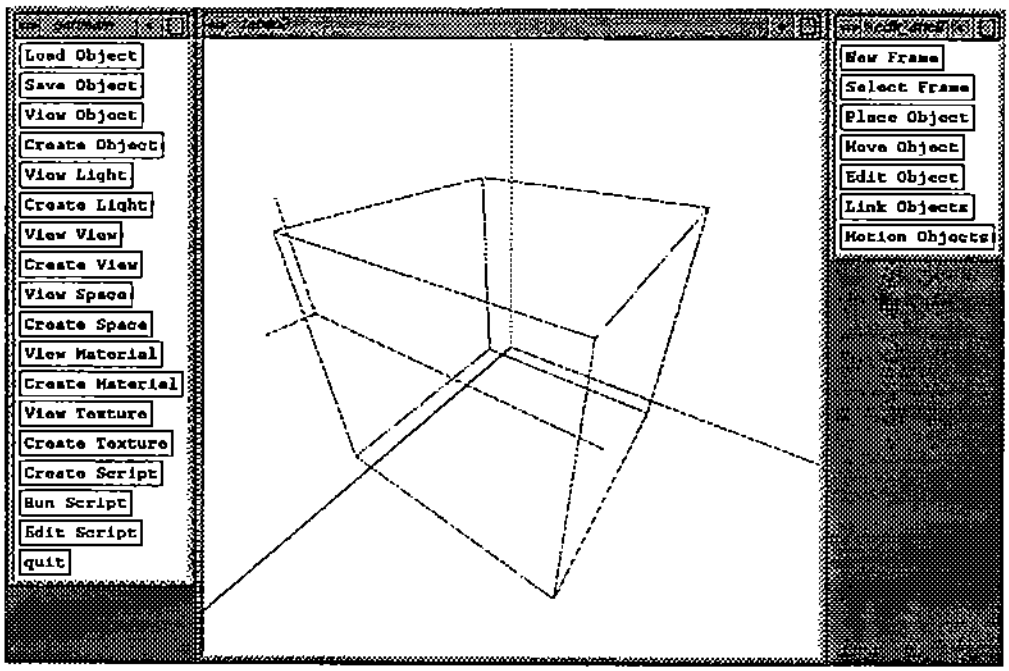

FIGURE 4: The GATI animation editor

4.4 Implementation Details

The GATI system has been implemented on a collection of unix workstations in a machine independent manner. It accomplishes this by using an extended version of the XS Graphics Libraries developed as a part of the SHASTRA project. The XS Graphics Libraries are a suite of 3D graphics libraries under X-11 that access system-dependent graphics facilities (and hardware) in a uniform, system-independent manner [1].

5 Examples

In this section we outline experiences with using GATI to implement some interactive animated applications within the area of geometric modeling and robotics.

GATI is being used in the development of an experimental algorithm for performing low cost obstacle avoidance, generating animations of families of implicit functions and an interactive application for manipulating the structure of molecules.

5.1 Motion Coordination

A recurring problem in robot motion planning is coordinating the motion of a collection of objects while ensuring that none of the objects collide with each other. GATI provides many benefits when used for visualizing this particular problem. It provides immediate feedback to changes in the motion planning algorithm. It also removes from the application developer the burden of writing routines and functions for manipulating graphical objects. Thus allowing her to concentrate upon solving the motion planning problem and not delving into the details of how to draw and animate graphical objects on a workstation.

We have implemented an experimental motion planner that relies upon GATI for input and display. The motion planner is a prototyping application for experimenting with various heuristic and deterministic algorithms for motion planning. An initial problem that we are experimenting with is given n objects in three dimensional space, each with an initial position and final position, compute paths for the objects such that no two objects collide as they travel along these paths.

Our current algorithm is as follows:

- For each object place it at its initial position and specify its final position in three dimensional space.
- A polynomial parametric surface is either calculated or specified by the user such that all of the initial points and final positions lie on that surface. (Even though the objects move in space, they follow restricted polynomial parametric trajectories and so naturally all their paths lie on a suitable polynomial parametric surface.)
- The parametric equations of this surface is then used to project the initial and final points on the surface to the parametric plane. Motion Paths are now computed in this parametric plane.
- A Voronoi diagram in the plane is next created from the initial positions of the objects.
- Initial paths are then generated from the objects start positions to there final positions.
- Now we perform the following loop:
- Calculate the minimum possible collision time between neighbors in the Voronoi diagram.
- Move all objects along their paths to this time.
- Update the Voronoi diagram and locally modify the paths of the objects that have a potential collision.
- repeat until all objects have reached their goal positions.

The GATI animation system has provided a great deal of benefit in the development of this application. Particularly useful was the immediate feedback that was provided. This help a great deal in the debugging process as it was visually obvious when the algorithm was performing incorrectly.

Figure 5 displays the paths of three moving objects from initial positions towards their goal positions moving in such a fashion to ensure no collisions.

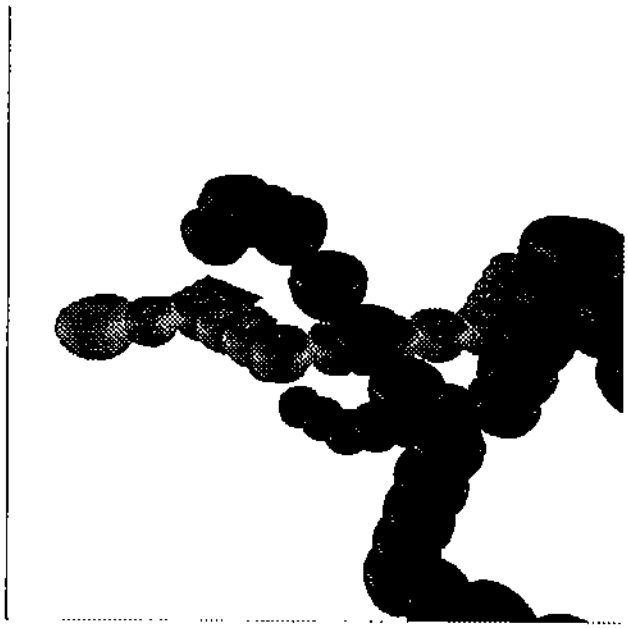

FIGURE 5: The paths of moving objects avoiding each other

5.2 Mathematical Animations

GATI was used to enhance the capabilities of an existing SHASTRA application called GANITH[3]. GANITH provides tools for visualizing algebraic surfaces and performing complex manipulations of these surfaces. GATI provided GANITH with a mechanism for animating the presentation of a sequence of members of a family of implicit functions.

In figure 6 an overlay of a sequence of steps in an animation shown. The picture is that of a family of concentric circles intersecting a quartic.

Figure 7 depicts a sequence of surfaces that smoothly join three pipes together. An animation of this type can be used to view and choose a surface that best joins three pipes and meets other visual criteria.

In figure 8 an overlay of snapshots of an animation depicting the intersection of families of surfaces is given.

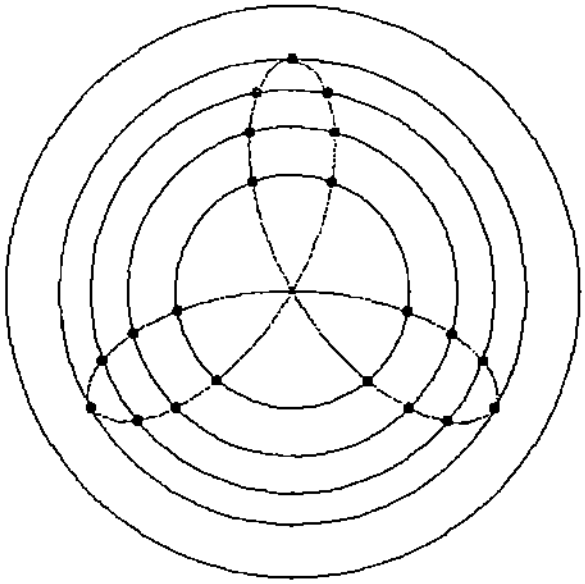

FIGURE 6: A family of algebraic curve intersections

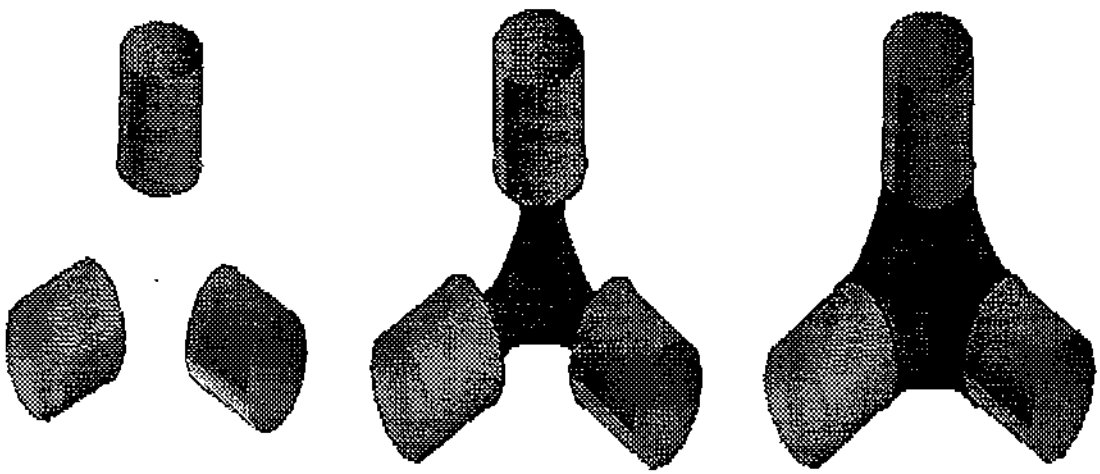

FIGURE 7: A family of smoothly joining surfaces

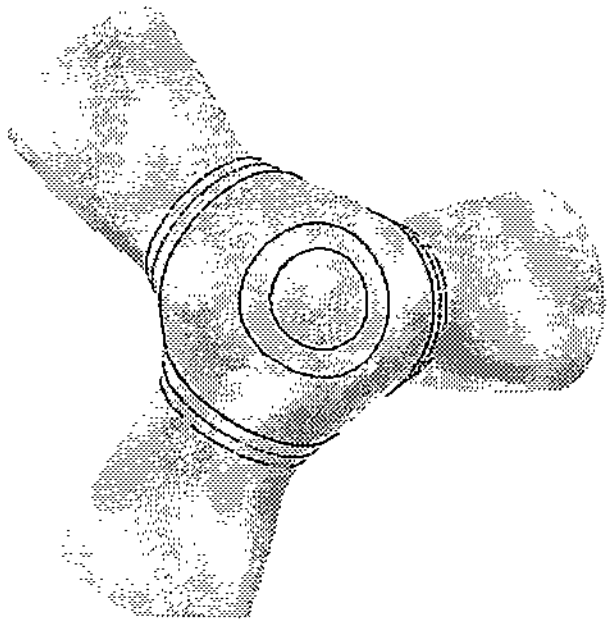

FIGURE 8: A family of intersecting surfaces with one surface shown

5.3 Molecular Docking Animation

An interactive application was developed using GATI to visualize the "docking" of drug and protein molecules under molecular Brownian motion.

The application reads in a description of the atom locations of a molecule from a file, computes the bonding information and then displays the molecule through GATI. Then either under program control (via the Gati Language) or a user can select two or more bonded atoms of the molecule. The angle between these bond atoms is varied with time and the structure of the molecule is updated and redisplayed. This provides for a animated view of how attractor drug molecules "dock" with protein molecules under dynamic situations.

6 Conclusions

In this paper we have presented GATI an animation system for providing low cost real-time interactive three dimensional animation services to computational geometry applications within the

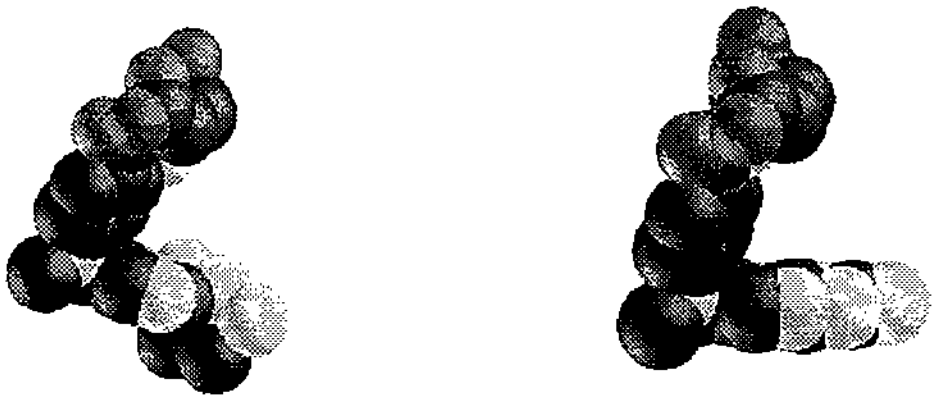

FIGURE 9: Two snapshots of an animated drug molecule

SHASTRA environment. GATI has many useful and novel features including the ability to create collaborative animations, animation events, and mixing two-dimensional and three-dimensional animations via the use of the spaces abstraction.

7 Acknowledgements

We thank Assumpta Sabater for her help in the development of the motion planner and Dr. Andrew Royappa for his assistance in the molecular docking animation project.

References

[1] V. Anupam, C. Bajaj, A. Burnett, M. Fields, A. Royappa, and D. Schikore. *XS: A Hardware Independent Graphics and Windows Library.* Computer Science Technical Report, CAPO-91-28, Purdue University, Department of Computer Sciences, 1991.

[2] V. Anupam, C. Bajaj, and A. Royappa. *The SHASTRA Distributed and Collaborative Geometric Design Environment.* Computer Science Technical Report, CAPO-91-38, Purdue University, Department of Computer Sciences, 1991.

[3] C. Bajaj and A. Royappa. *The GANITH Algebraic Geometry Toolkit. Proceedings of the First International Symposium on the Design and Implementation of Symbolic Computation Systems, Lecture Notes in Computer Science*, 1990.

[4] M. H. Brown. Algorithm Animation. PhD thesis, Brown University, 1987.

[5] Stéphane Chatty. *Defining the dynamic behavior of animated interfaces. In 5th IFIP Working Conference on Engineering for Human Computer Interaction*, 1992.

[6] Alias Research Inc. 110 richmond St. East, Suite 500, Toronto, Ontario, Canada m5c-1p1.

[7] Symbolics Inc. 1401 Westwood Blvd, Los Angeles, CA 90024.

[8] Vertigo Systems International Inc. 119 W Pender St., Suite 221, Vancouver, BC, Canada v6b 1s5.

[9] Wavefront Technologies Inc. 530 East Montecito, Santa Barbara, CA 93101.

[10] Paul M. Isaacs and Michael F. Cohen. *Controlling Dynamic Simulation with Kinematic Constraints, Behavior Functions, and Inverse Dynamics. ACM COMPUTER GRAPHICS (Siggraph Proc. '87)*, 1987.

[11] N. Magnenat Thalmann and D. Thalmann. Computer Animation Theory and Practice. Springer Verlag, 1985.

[12] Abel Image Research. 953 N. Highland Ave., Los Angelese, CA 90038-2481.

[13] C. Reynolds. *Computer Animation with Scripts and Actors. Computer Graphics (Siggraph)*, 1982.

[14] John T. Stasko. *TANGO: A Framework and System for Algorithm Animation. Computer*, 1990.