A CONTACT DETECTION AND MOTION MODIFICATION ALGORITHM FOR TELEROBOTIC TRAINING

Susan Mauzy
NASA, Johnson Space Center
Houston, Texas

Bob Martella
CAE-Link Corporation
Houston, Texas

ABSTRACT

The Space Station Training Facility (SSTF) is an environment to train astronauts for Space Station Freedom (SSF) operations. Like most simulators, the SSTF includes an image generator (IG) that visually simulates the out-the-window scene and the closed circuit TV views of SSF. Because the displayed objects are graphic images, they can merge and violate their abstraction of solid objects. Most IGs provide the capability to detect collisions between predefined points and/or volumes. The application program sends an object position to the IG, the IG determines if the object has collided with other predefined objects, reports to the application and draws the object in its new position. The problem is in the delay associated with the IG processing time to compute the contact position and draw the object. By the time the IG reports the contact point to the application, the object has already merged with another object. For most applications this slight merging of objects is not perceivable to the student since normal viewing distances are large compared to object size and amount of overlap. For telerobotic training, the eyepoints can be so close that object merging can cause negative training. One solution is to check for contact before the application sends the object's position to the IG. This requires precise geometric models of the end-effectors and their workpieces to reside in the host computer. Additional computing resources are also required to run the contact detection algorithm. Rapid prototyping was used to verify the solution approach and reveal limitations in the design. This paper presents the contact detection and motion modification algorithm and discusses its capabilities and limitations.

ABOUT THE AUTHORS

Susan Mauzy graduated from Rice University, Houston, Texas, with a BSEE in 1985 and a MEE in 1991. She has worked for NASA since 1985, working first as a programmer on the Single Systems Trainers for the Shuttle program and now as the Subsystem Manager for the Robotics and Environment systems for the SSTF.

Bob Martella attended the U.S. Air Force Academy and Virginia Tech where he received a BSME in 1983. He has worked for the CAE-Link Corporation, Houston Operations since graduation. He began his career at Link modeling propulsion systems for military applications and has since worked modeling avionics systems, motion systems, and various other aircraft systems. He is presently a Systems Engineer working in the Robotics Group for the SSTF.

A CONTACT DETECTION AND MOTION MODIFICATION ALGORITHM FOR TELEROBOTIC TRAINING

Susan Mauzy
NASA, Johnson Space Center
Houston, Texas

Bob Martella
CAE-Link Corporation
Houston, Texas

Introduction

The Space Station Training Facility (SSTF) will model all onboard systems for the purpose of training astronauts and mission controllers. The astronaut training will emphasize the interaction between the operation of different systems. The mission controller training will emphasize teamwork in the ground control of the SSF. The SSTF will receive uplink commands and send simulated downlink data to the Space Station Control Center (SSCC) to support mission controller training.

The SSTF will model the Mobile Servicing System (MSS). The manipulators included in the MSS are the Mobile Transporter (MT), the Space Station Remote Manipulator System (SSRMS), and the Special Purpose Dexterous Manipulator (SPDM). The MT will act like a motorized flatbed rail car to transport the other manipulators along one face of the truss. The SSRMS is a long (fifty-four feet), strong arm with seven degrees of freedom (DOF). The SSRMS will move objects for assembly and maintenance, berth the Orbiter, and position the SPDM and astronauts. The SPDM is an anthropomorphic telerobot with two seven DOF arms mounted on a five DOF base and body. The SPDM will support or replace astronauts in the performance of Extra-Vehicular Activity (EVA) maintenance tasks.

The SSTF must realistically model the mechanical interfaces between the MSS and other SSF systems where the crew is responsible for controlling the connections. The sole interface mechanism for the SSRMS is a Latching End Effector at each end. These LEEs will attach to standard grapple fixtures. Typically, one LEE will attach to a standard grapple fixture and act as a base while the other LEE attaches to a standard grapple fixture on the object to be moved. The SPDM has three mechanical interfaces. A LEE and a standard grapple fixture on the SPDM's base provide alternative methods for supporting the SPDM at a worksite. The SPDM arms each have a versatile end-effector called the Orbital Replacement Unit (ORU) Tool Changeout Mechanism (OTCM). This device can grapple small, specially-shaped knobs. The OTCM can also manipulate and operate a variety of rotary power tools.

The SSF crew will control the SSRMS and SPDM from workstations inside the pressurized volume. The workstations will receive crew input through hand controllers, a keyboard, and a panel of switches and dials. The crew members will monitor their progress through CCTVs, out-the-window viewing, and digital readouts in the control panel. The SSTF will provide functionally equivalent workstation hardware in the crew station mockups. An image generator will provide CCTV and out-the-window scenes.

One key aspect of realistic simulation is the response time of the simulated systems. The SSTF is

required to provide responses within 150 milliseconds of the real world SSF response time. This poses strict time constraints on the computational activities and necessitates innovative techniques to achieve the required performance. This paper describes one innovative technique for realistically simulating the mechanical interfaces of the MSS.

Problem Statement

An important aspect of training crew members in robotic operations is to give the student accurate feedback on the limitations of motion of the robots. In many cases it is sufficient to merely give the student a cue when a collision occurs. An example of this is when the elbow of the SSRMS is inadvertently commanded to move into the Lab Module, a warning tone sounds that cues the student that the SSRMS has collided with something. For training purposes it is not necessary to stop the motion of the SSRMS when unintentional collisions occur.

Conversely, it is necessary to stop the motion of the robot simulation when intentional collisions occur. Intentional collisions include collisions between the LEE of the SSRMS and a PDGF, and the OTCM of the SPDM and any of its tools or workpieces. Training for these activities requires a more realistic simulation. In this situation it is necessary to constrain the motion of these objects so that they do not pass through each other.

Figure 1 depicts the data flow path for commanding the robot arm using the contact detection capabilities of the IG. The hand controller inputs come from

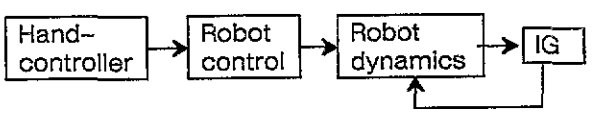

Figure 1: Robot Data Flow Path

the student station and are sent to the robotic control system which commands robot motion. Robot dynamics responds to these commands by computing the new position of the robotic arm and sending it to the IG. The IG then displays the new position and checks for volume collisions. Finally, it sends the collision information back to the robot dynamics. A time delay of approximately 100 milliseconds occurs from the time robot dynamics sends the position data to the IG to the time robot dynamics receives the collision information from the IG. Because of this delay, volumes can pass through other volumes. This effect is referred to as image ghosting. Robot camera proximity makes this effect very pronounced through CCTV.

Since the SSTF training focus is systems training and not specifically Robotics training, the intentional interactions between robot arm and workpieces have the highest priority. Thus while ghosting is acceptable for inadvertent collisions, it is not for planned interactions between manipulator and ORUs.

Solution

Design Approach

One solution is to determine when contact occurs before the IG displays the scene. Figure 2 shows the new data flow path with contact detection included.

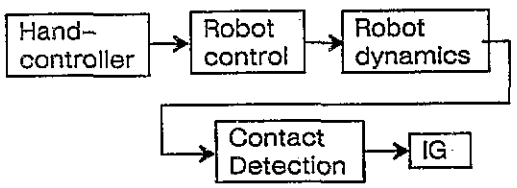

Figure 2: Modified Robot Data Flow Path

This routine is invoked after the robot dynamics subroutine is executed. The new algorithm checks the geometries of the objects in question to determine when intersections occur. When the check finds that contact has occurred, the commanded motion is not allowed to move the object. The invalid position vectors are not reported to the IG, eliminating the delays associated with IG collision checking routine. The effect is that the end effector does not respond to commands that will cause objects to ghost. This results in positive, realistic feedback to the trainee. This solution approach was prototyped for SSTF in Ada using object oriented concepts on 2 Silicon Graphics 220 VGX workstations.

Data Structure

Each rigid body is considered a separate object that is composed of many polyhedrons. The polyhedrons are structured in a tree. A polyhedron is a variant abstract data type. It can be either a subtree or a leaf. If it is defined as a subtree then it has access to two other polyhedrons: the left_child and the right_child. If it is defined as a leaf, then it contains data that describes a hexahedron. Figure 3 illustrates this tree structure.

The hexahedron data structure consists of various pointers (access types) that organize the vertices into edges and faces. This structure facilitates the calculations of the line equation coefficients and the plane equation coefficients that define the edges and faces

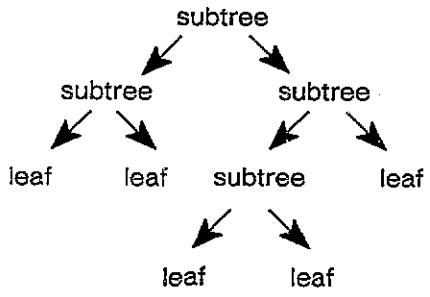

Figure 3: Object Tree Structure

of the hexahedron. The line coefficients are shown in Equation (1A), the equations of a line in three-dimensional space.

$$\frac{X - X_0}{E} = \frac{Y - Y_0}{F} = \frac{Z - Z_0}{G} \qquad (1A)$$

Equation (1B) shows the parametric form of the line equations with the parametric parameter t:

$$X = X_0 + E\,t$$

$$Y = Y_0 + F\,t \qquad (1B)$$

$$Z = Z_0 + G\,t$$

The plane coefficients are from the general plane equation shown in Equation 2.

$$AX + BY + CZ = D \qquad (2)$$

The hexahedron data also contain local and global vertex positions. The local vertices are defined in the local coordinate frame of the hexahedron. Figure 4 illustrates the hexahedron and its coordinate frame.

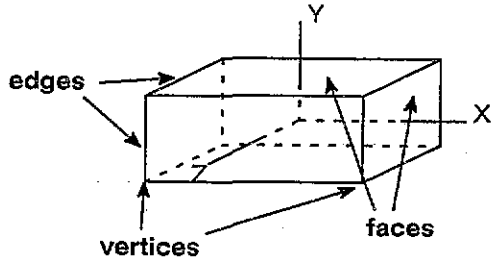

Figure 4: Hexahedron

The local vertices are used to calculate the position of the vertices in the global coordinate frame when the object is in motion. The global vertex positions are used to calculate the line equation coefficients and the plane equation coefficients.

All polyhedrons have a commanded position, an actual position, a tree position, and a radius. The commanded and actual position are in global coordinates. The tree position is in the root polyhedron coordinate frame. The radius of a polyhedron is defined as the distance from the coordinate frame center of the polyhedron to the most distant vertex. Figure 5 contains the Ada that defines the data structure.

Theory of Operation

A rigid body is either moving or stationary. Only moving objects can collide with other objects. If an object is commanded to move, that object must determine if it will collide with another object before it moves to that space. This involves modeling the geometry of each object of interest in three dimensions and testing for intersections between moving objects and other objects. If the commanded movement results in an intersection of two polyhedrons, the commanded motion is ignored and the object is simply not moved to that position. Only motion that does not cause a collision is allowed. If no contact is detected, the actual position of the object is set to the commanded position, the IG is notified, and the image of the object is seen in its new position in the visual.

Each moving object is checked to determine if it is commanded to collide with another object. This is accomplished by a coarse check and a fine check. The coarse check is a very fast check that determines if there is a possibility that two objects are in contact. If the coarse check fails any hexahedrons, the fine check must be performed on them. The fine check is computationally more intense. It computes the exact positions of the hexahedrons in order to calculate the intersection (if one exists).

Algorithms

The Contact Detection Buhr diagram is shown in Figure 6. It illustrates the interaction of the subprograms inside the Contact Detection model.

Find_global_position is a recursive procedure that calculates the commanded position of each hexahedron in each moving object's tree structure. The orientation elements (roll, pitch, yaw) of 6x1 position vectors can not be explicitly added together. They must be decomposed into their incremental effects on each axis and then added. This is accomplished by converting the commanded 6x1 position vector of the object and the 6x1 tree position vector of the hexahedron into 4x4 homogeneous transformation matrices.[1] The matrices are multiplied and the result is converted back to 6x1 position vector. This is the commanded position of the hexahedron in the global coordinate frame.

```
type LINE_COEFS is (XO, YO, ZO, E, F, G);
type LINE_COEF_TYPE is array (LINE_COEFS) of REAL;
type PLANE_COEFS is (A, B, C, D);
type PLANE_COEF_TYPE is array (PLANE_COEFS) of REAL;

type VECTOR_3X1 is array (1..3) of REAL;
type VECTOR_3X1_POINTER is access VECTOR_3X1;
type VECTOR_6X1 is array (1..6) of REAL;

type EDGE_TYPE is array (1..2) of VECTOR_3X1_POINTER;
type FACE_TYPE is array (1..4) of VECTOR_3X1_POINTER;
type ALL_VERTICES_TYPE is array (1..8) of VECTOR_3X1;
type ALL_VERTICES_POINTER is array (1..8) of VECTOR_3X1_POINTER;
type ALL_EDGES_TYPE is array (1..12) of EDGE_TYPE;
type ALL_FACES_TYPE is array (1..6) of FACE_TYPE;
type ALL_LINE_COEFS_TYPE is array (1..12) of LINE_COEF_TYPE;
type ALL_PLANE_COEFS_TYPE is array (1..6) of PLANE_COEF_TYPE;

type HEXAHEDRON is
    record
        LOCAL_VERTICES      : ALL_VERTICES_TYPE;
        GLOBAL_VERTICES     : ALL_VERTICES_POINTER;
        EDGES               : ALL_EDGES_TYPE;
        LINE_EQ_COEFS       : ALL_LINE_COEFS_TYPE;
        FACES               : ALL_FACES_TYPE;
        PLANE_EQ_COEFS      : ALL_PLANE_COEFS_TYPE;
    end record;

type NODE_TYPE is (LEAF, SUBTREE);
type POLYHEDRON (NODE_DESCRIPTION : NODE_TYPE := SUBTREE);
type POLYHEDRON_LIST is access POLYHEDRON;
type POLYHEDRON (NODE_DESCRIPTION : NODE_TYPE := SUBTREE) is
    record
        POS_ACT             : VECTOR_6X1    := (others => 0.0);
        POS_COM             : VECTOR_6X1    := (others => 0.0);
        TREE_POSITION       : VECTOR_6X1;
        RADIUS              : FLOAT          := 0.0;
            case NODE_DESCRIPTION is
                when LEAF =>
                    DATA            : HEXAHEDRON;
                when SUBTREE =>
                    LEFT_CHILD      : POLYHEDRON_LIST;
                    RIGHT_CHILD     : POLYHEDRON_LIST;
                end case;
    end record;
```

Figure 5: Data Structure for Contact Detection

Polyhedron_in_range is the coarse check. It compares the sum of the radii of the two objects in question and the absolute distance between the object centers. If the distance between the coordinate centers is less than the sum of the radii of the two objects, then there is a possibility that the two objects are in contact. The routine recursively searches the tree structures of the objects. It returns a linked list of moving and static object pairs that failed the radii condition. If at least one possible contact pair is returned, the fine check must be performed. If there are no possible contact pairs encountered, the commanded motion is allowed.

When the contact_pair list is not empty, the fine check is performed to determine if the two hexahedrons are in contact. The two procedures Calculate_equations and Find_contact_points accomplish this check.

Calculate_equations is executed first. It modifies the edge and plane equation coefficients of the moving hexahedrons from the present position to the commanded position. This procedure transforms the commanded position vector into a matrix. This matrix is used by **Transform_points** to update the vertices of the hexahedron with the commanded position. The

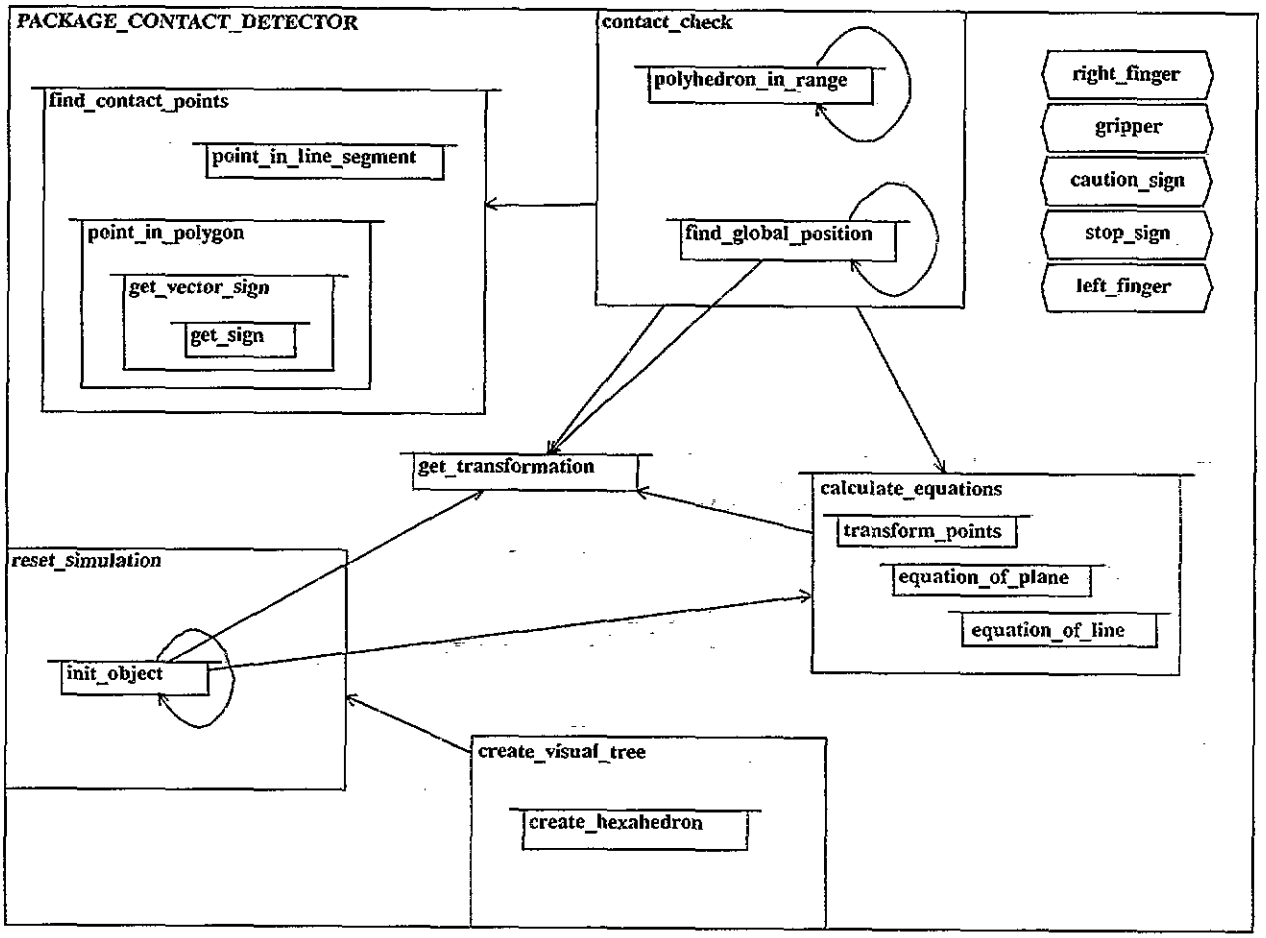

Figure 6: Contact Detection Buhr Diagram

Equation_of_plane procedure then calculates the coefficients for the equation of the plane that encloses each face. The equation of the plane is shown in Equation (2).

The coefficients A B and C are found from the normal vector to the plane. Three of the four vertices that define a face are used to create two vectors. The cross product of the two vectors results in a vector normal to the plane. The vertices and the vectors are chosen in such a way that the normal vector has direction away from the centroid of the hexahedron. The elements of the vector (X, Y, Z) are equal to the first three plane coefficients (A, B, C). The final plane coefficient is calculated by substituting one vertex into the plane equation and solving for D.

The final subroutine in Calculate_equations is **Equation_of_line**. It returns the coefficients of the line that encloses each edge. The equation of the line is given in Equation (1). The parametric parameter t is arbitrarily assigned a value associated with each vertex that defines the edge. For the first vertex, t = 0; for the second, t = 1. This simplifies the calculation for the equation of the line.[2]

Find_contact_points traverses the list of contact pairs to determine if any are valid. It first solves for the intersection (if one exists) of each line and each plane for each hexahedron pair. It then imposes the boundary conditions of the edge and the face with the routines Point_in_line_segment and Point_in_polygon respectively.

Point_in_line_segment determines if the point of intersection is within the line segment defined by an edge of a hexahedron. This check is accomplished by inputting the point of intersection into the parametric equations of the line and solving for the parametric parameter. Since the parametric equations were defined using the the two vertices of the edge, the parametric parameter must be between zero and one, in-

clusive, for the intersection point to be on the line segment.

Point_in_polygon determines if the point of intersection is within the polygon defined by a face of a hexahedron. This is accomplished using vector cross products and the right hand rule. Beginning with an arbitrary vertex, a first vector is created from the vertex to the point of intersection and a second vector is created from the vertex to the next vertex. This is done for each vertex of the polygon in an arbitrary (CW or CCW) direction around the perimeter of the polygon. If the sign of each vector product is the same, then the point resides within the polygon. If a change of sign occurs between any of the vectors, then the point of intersection is not on the face of the hexahedron and the particular face and edge are not in contact.

The following two procedures are not executed in real-time, they are used for setting up the simulation for execution.

Create_visual_tree is a constructor operator. It constructs the tree structure for each object and initializes the data in each tree. It is executed only when the simulation is started. When Create_visual_tree builds a leaf polyhedron, it utilizes **Create_hexahedron** to assign the data. This function creates the access types, assigns the data, and returns the hexahedron given its length, width, and height.

Reset_simulation is also a constructor operator. It initializes the position of each object defined in the simulation. The subprogram **Init_object** is called for each tree structure. It sets the position of each object to its initial position, and then makes a call to Calculate_equations to calculate the correct coefficients. Reset_simulation is executed only when the simulation is first started or when it is reset without terminating execution.

Limitations

This software is the result of a rapid prototyping effort using object oriented methodologies in Ada. Requirements were generated to drive the scope of the prototyping effort. The system was designed, coded, and tested. This process has revealed limitations that can be traced to the lack of good requirement descriptions and/or a desired increase in the functionality of the system. These limitations are being addressed in the second build of Contact Detection.

Execution time is always a priority in real-time simulation. This design requires memory to be allocated to create the contact_pair list with each iteration because the length of the list is unknown. This dynamic memory allocation of (and destruction of) the contact_pair list is too time consuming for real-time constraints.

Packaging of all objects are defined in one package. This requires that the Contact Detection package be updated whenever new objects get created. This reduces modularity and reusability of the software. Further, since the moving of the object was essentially performed within the Contact_check procedure, any additional objects would cause an update to not only the body but also the specification.

Selectability of objects is not available for this design. It would be desirable that the instructor have the option to select or deselect individual objects for contact checking. Further the instructor may want to activate or deactivate all contact checking at once. These features are not supported in this design.

Latching of objects is not addressed in this design. Since all objects are hard coded into the contact detection package, there is no facility to latch objects together other than hard code a latch to particular objects. This is inefficient and dangerous.

Dynamics associated with bumping an object or pushing an object is not addressed with this design. This effect would require a higher fidelity routine with force calculations for each link at a minimum. Examining the case of attempting to push an object with the end effector, some joints may not have sufficient torque to move while others do, resulting in uncommanded motions. If the object is in motion it may also backdrive the joints in the arm.

Current activities

Contact Detection is currently going through a second build process. The first build of the contact detection algorithm has revealed the above limitations which might not have been foreseen in a standard waterfall development process. The results of the prototyping activity have fuelled this incremental build design philosophy. The results are evident. Working software is developed early in the program life cycle, performance capabilities become very evident, and risky areas are addressed, all before integration occurs.

Execution time

A major goal of the second build is reducing the execution time. The intermediate step of creating a contact_pair list will be deleted. When a possible contact pair is encountered the algorithm will immediately

determine if contact has really occurred before it continues to look for the next possible collision pair. As a result execution time will not be wasted with dynamic memory allocation required for the linked list.

Packaging

The packaging for Contact Detection will be redesigned in the second build. The first build design had all contact objects existing within the package Contact_detection. The second build will allow anyone to create and use a contact object wherever he chooses. The package Contact_object will contain the attributes of any contact object and the functions that can be performed on it. Users will construct objects from that template and use them in accordance to their defined functionality. The construction process includes a registration routine. This routine registers each object with the Object_manager. The Contact_manager will manage the list of Contact_objects and provide access to the objects upon request.

Selectability

The second build is also adding a selectability function for one or all of the objects. This feature will allow the instructor to turn the Contact Detection function on or off without affecting the rest of the simulation. The instructor will also have the capability to selectively change the objects that are being checked for collisions without stopping or freezing the simulation.

Latching

Contact Objects will have an additional function of latching. This function will allow a object the capability to move another object that it is latched to. Thus objects can be passive or both passive and active.

Dynamics

Modelling the dynamics of object collisions requires the modelling of the forces imparted between the objects. This is not currently a priority in the contact detection design since the robotic simulation in the SSTF does not have the requirement to model to this fidelity. There is the possibility that this capability will be added to the SSTF in the future. For this reason the contact detection model is being developed with maintainability, modularity, and upgradeability in mind.

Summary

The prototyping effort discussed in this paper describes one solution to the problem of having one image pass through another in a visual display. The IG itself can not prevent this quickly enough to be undetectable to the human eye, so the computer calculating the movement of the object must perform this function. The algorithm already developed and tested does have some limitations, but we are working to overcome these to provide a viable method for contact detection and motion modification in a real-time simulation using object oriented techniques. The routines developed here are general enough to be applicable in a variety of problems where an image generator separate from the simulation host is used to display objects that can collide with each other.

Acronyms

CCTV	Closed Circuit Television
DOF	Degree Of Freedom
EVA	Extra-Vehicular Activity
IG	Image Generator
LEE	Latching End Effector
MSS	Mobile Servicing System
MT	Mobile Transporter
ORU	Orbital Replaceable Unit
OTCM	ORU/Tool Changeout Mechanism
PDGF	Power/Data Grapple Fixture
SPDM	Special Purpose Dexterous Manipulator
SSCC	Space Station Control Center
SSF	Space Station Freedom
SSRMS	Space Station Remote Manipulator System
SSTF	Space Station Training Facility

References

1 Paul, R.P. Robot Manipulators: Mathematics, Programming, and Control, The MIT Press, Cambridge, Massachusetts, 1981.

2 Swokowski, E.W., Calculus with Analytic Geometry, Prindle, Weber & Schmidt, Inc., Boston, Massachusetts, 1975.