Simulating Physical Constraints during Virtual Assembly using Constrained Motion

S. Jayaram, H. Chandrama, Y. Wang
School of Mechanical & Materials Engineering
Washington State University
Pullman, WA

K. Lyons
Manufacturing Systems Integration Division
National Institute of Standards and Technology
Gaithersburg, MD

In recent years, significant attention is being focused on the assembly process during the design phase of a product. Due to increased complexity, these products must also be thoroughly analyzed for “productibility” and “maintainability” before committing the high capital required to produce physical prototypes. 3D design and manufacturing software tools can greatly assist in this analysis. The complete integration of these software tools is an important goal of the designers and creators of C4 (CAD/CAM/CIM/CAE) systems. Achieving this integrated 3D engineering process capability will provide a means to refine and develop a product or process with significant saving of cost and time. It is envisioned that the integration of Virtual Reality (VR) with software systems for engineering, design, and manufacture will provide a new boost to the field of Computer-Aided Engineering. In particular, one aspect that will be affected significantly by VR is the use of Virtual Assembly for designing products.

“Virtual Assembly” (VA) is defined as: “The use of computer tools to make or assist with assembly-related engineering decisions through analysis, predictive models, visualization, and presentation of data without physical realization of the product or supporting processes.” VA is a combination of several technologies such as advanced visualization and virtual reality, simulation, decision theory, and assembly and manufacturing procedures.

A large portion of a product’s cost can be associated with field-related activities (e.g., warranties, field replacement units) and increased attention is being focused on these life cycle considerations. To address assembly and maintainability concerns of a system/subsystem earlier
in the product realization process, researchers have demonstrated techniques to define the assembly and disassembly trajectories. This will allow for the creation of swept volumes that will depict the volume (soft zone) of space that the component travels through during assembly (or removal process required for maintenance). The trajectory information can also be sent to a Design for Assembly (DFA) system to allow an engineer to suggest design changes which could be fed back to the design process. Thus, it is important that the data and information generated by VA systems be extremely accurate. The information from virtual assembly systems can be used for 1) robot path planning, 2) process planning systems, or 3) training personnel for the assembly process. Connacher et al. (1995) have outlined the issues and benefits related to the use of virtual reality for virtual assembly.

Most automated assembly planning systems function without the designer or include the designer in a non-intuitive fashion. To overcome some of the shortcomings of algorithmic assembly planning, intuitive designer interaction is required. An important issue related to intuitive designer interaction within virtual worlds is the lack of physical constraints that the user depends on in the real world. To compensate for this, the user needs to be provided with constraints required for precisely manipulating virtual objects.

If the process information generated by VA systems is to be utilized by manufacturing engineers for serious applications, constraints need to be implemented to augment interaction within the virtual environment. This is warranted by the following considerations: 1) humans lack the fine motor control necessary to manipulate objects precisely, and 2) virtual worlds lack haptic feedback which we depend on so much in the physical world. The absence of constraints when interacting with virtual objects is a major limitation in current virtual reality applications.

This paper presents the methods for “mapping” the motion of a component in 3D space to a constrained space in the virtual environment. The information about the constraint relationships between assembled parts/sub-assemblies is extracted from a CAD system. This information is used during the virtual assembly process to imitate the physical world operations.
Literature Review

The main areas of interest in this literature review are the following: 1) incorporation of geometric constraints, 2) creation of swept surfaces and volumes when performing assembly operations, 3) real-time collision detection methods, and 4) haptic feedback.

As stated by Fernando et al. (1994), "a common weakness of the existing virtual environments is the lack of efficient geometric constraint management facilities such as run-time constraint detection and the maintenance of constraint consistencies during 3D manipulations." Their paper describes a constraint management facility that uses automatic constraint recognition to recognize such constraints as against, coincidence, tangency, and concentricity is described.

Mine (1997) has described the implementation of virtual constraints into the ISAAC system. Virtual constraints are those in which the application ignores the extraneous degrees of freedom in the user's input data. ISAAC is a scene composition application used for the interactive construction of virtual worlds. One of the goals of ISAAC is to incorporate constrained manipulation modes and grid functions that will help in the precise manipulation of objects. ISAAC includes three types of constrained motion widgets for the constrained manipulation of objects: one-dimensional translation, two-dimensional translation, and one-dimensional rotation.

Swept surfaces and volumes are generated by an object as it moves through time and space in an arbitrary, time-dependent trajectory. Schroeder et al. (1994) have applied this concept to the problem of maintainability of jet aircraft engines and "safe" path planning in robot applications.

Martin and Stephenson (1990) present a method to evaluate the volume swept by a 3D object as it moves along an arbitrary path. Envelope theory from differential geometry is used to find the volumes swept out by the individual surfaces of a solid body, and an algorithm is presented to show how the results of sweeping the individual surfaces can be combined to form a new 3D model of the swept volume.

The R&D center at General Electric has developed a software that can automatically determine if an access path is available for a given design. The paper by Chang and Li (1995) explains some of the applications of this software. The software, Product Vision (PV), works with data sets from conventional CAD/CAM software to calculate an access path for a given part
and displays this path within the 3D animation environment provided by the PV software. Human manipulation of the parts has not been implemented with this software.

Cohen et al. (1995) present an algorithm for exact collision detection in interactive environments. A two-level hierarchical collision detection system is presented which selectively computes the precise contact between objects in a multi-body environment. These methods have been implemented in a system called I-COLLIDE which has been designed for use in large environments composed of convex polytopes. At a fundamental level I-COLLIDE takes advantage of the coherence that exists from frame to frame to efficiently determine when a pair of objects are colliding.

Smith et al. (1995) present another collision detection algorithm for use in virtual reality applications. The algorithm can be used directly for both convex and concave objects and objects can be deformed during motion. It works efficiently by first reducing the number of face pairs that need to be checked accurately for interference by first localizing possible collision regions using bounding box and spatial subdivision techniques.

Buttolo and Hannaford (1995) describe the structure of a force display implemented for precise manipulation of scaled or virtual environments. A force display is a manipulator designed to provide and receive kinaesthetic information to/from a human operator. The human operator interacts with this pen-based force display using a pencil or a scalpel. Such a system is effective for micro-surgery simulation applications.

Burd ea et al. (1995) have presented the second generation Rutgers Master (RM-II), which integrates position-sensing and force feedback to multiple fingers in a single structure, without the use of sensing gloves. Once again, most applications for this device have been in the area of medical training.

**Problem Statement**

When performing assembly in a virtual environment, the user is able to move parts through other objects and perform many other operations which may be physically impossible in the real world. To model real world operations more accurately in virtual assembly environments, objects should move as expected when pushed, pulled, or grasped. Following are
some of the issues which need to be considered when performing assembly operations in real-time virtual environments.

Accurate collision detection is required for any degree of realism. In modeling collision detection there are several cases which are difficult to model - e.g. colliding objects that do not interpenetrate, or two objects that exert a force on one another. With thousands of objects in a complex virtual environment, and intense computation requirements, it would take an extremely long time to check for all possible collisions.

Haptic feedback is required to facilitate manipulation of objects. However, it is not easy to define a mathematical model that would allow real-time simulation for haptic interfaces with sufficient stability for the full range of possible responses - from rigid bodies to penetrable objects.

Virtual reality hardware devices, especially tracking devices, are difficult to use and some form of calibration is required in most environments. The generation and importing of soft zone data into the CAD system will provide the designer with much-needed information on the assembly operation. For the creation of soft zones, it would be critical to eliminate the unnecessary information generated due to the shakiness/jitters of the human hand or inaccuracies of the tracking and sensing devices.

Tolerancing is one of the essential elements for meeting functional requirements of virtual assembly. When evaluating tolerance issues within virtual environments it is all the more important for the generated information to be precise.

These issues and obstacles highlight the need for a method to simulate physical constraints of the real world, without sacrificing ease of user interactivity. In a real assembly process, people use physical constraints extensively in assembling the parts. Examples of these include sliding a rod through a hole, sliding a block along a groove, or mating surfaces and then aligning holes. Absence of constraints in virtual assembly environments is a major limitation and some of these must be included as geometric constraints in the virtual environment.

**Proposed Solution**

The importance of including constraints and constrained motion in a virtual assembly system makes itself most apparent when assembling an object in a virtual environment without
any such restrictions. A user can move a part through other objects to its final location and perform other non-intuitive assembly operations. A case study was performed consisting of a manual assembly operation of average difficulty. It was noticed that many of the constraints used in actual physical assembly of components are applied automatically and intuitively from real world experience. For example, when assembling a nut on a bolt, the alignment of the axes and the mating of appropriate planes is automatically performed. This type of intuitive information is of primary importance for a successful implementation of a virtual assembler.

In the case study, after performing the manual assembly, a similar operation was performed in a CAD environment using Pro/ENGINEER, keeping the physical operations in mind. A detailed analysis indicated that operations in the manual assembly are comparable to those performed in Pro/Engineer. For the virtual assembler, the main task is to achieve the desired result of a realistic sequence of assembly operations by constraining the part’s motions in space. Assembly constraints on a part or sub-assembly serve no purpose unless the motion of that part or sub-assembly is constrained during assembly to imitate the physical world operations.

The solution proposed in this paper is to map the actual motion of the constrained part to a constrained space. Many constraints in real-world assemblies involve either axial alignment or surface mating with or without offsets. For the axial case, when the conditions which satisfy a given constraint are met, the part is restricted to move only along that axis. Thus, the only movements allowed are translation along the axis and rotation about the axis. When considering planar constraints, motions are restricted to movement along the plane specified and rotation about the normal vector of the plane.

The information about the constraint relationships between assembled parts/sub-assemblies needs to be extracted from the CAD system. This consists of the 1) type of constraint (align or mate), 2) geometry of the constraint, and 3) offset information defining the constraint. A description of how the constraint information is extracted can be found in the paper by Connacher et al. (1996). The constraint information is used in the work described in this paper.

**Virtual Assembly Design Environment (VADE)**
A prototype “Virtual Assembly Design Environment” (VADE) has been defined and implemented at Washington State University to address a specific assembly scenario representative of actual issues facing an industry assembly facility (Connacher et. al, 1997). This consists of a virtual environment which allows an engineer to consider assembly issues early in the design cycle. The virtual environment consists of virtual reality hardware and software, which allows the designer to be immersed in the environment. A stereo-scopic head-mounted display is used to provide the engineer with graphics feedback. Electromagnetic positioning devices are used to track the head and hand movements of the user. This enables the user to “look around” and manipulate a model of the hand in the environment. The CyberGlove™ is used to monitor the movements of the fingers and the wrist. Thus, the right hand model is fully dexterous.

Each product has a “base part” or a host part on which the assembly is built. Parts and/or sub-assemblies of a given product are modeled separately and can be merged into the base part to generate the final assembly. The design of the base part decides the jigs and fixtures that may be used to support it during the assembly process. Once in the environment, the user will notice the “base part” held in the left hand at the location and orientation of the electromagnetic tracking device and a desk containing all of the remaining parts for assembly arrayed in “bins” for easy selection (Figure 1). To select a part for assembly, the user moves the right hand into the bin containing the part and grasps the part. Gripping is based on the number of contacts, direction of contacts, and a friction factor. This allows realistic “grabbing” of the part. More details about this gripping method can be found in Connacher (1996). The part can then be manipulated in 3D space and moved to its final location and orientation for assembly. During this process, the base part can also be manipulated with the left hand. The user then selects the next part and continues with the assembly process. The engineer will be able to use this virtual assembly environment to evaluate tolerance issues, generate assembly/disassembly path and process plans, select optimal component sequencing, and visualize the results.
Fig. 1. **Visual environment of the VADE system.**

Performer™ (developed by Silicon Graphics, Inc.) is used to generate the graphics for this system. Performer™'s high-level graphics library allows developers to separate themselves from the intricacies of lower-level graphics programming and concentrate on the application. In Performer™, a scene is created by generating a hierarchy of "nodes", attaching these nodes to the scene, attaching the scene to a channel, and finally attaching the channel(s) to a graphics pipeline. The scene is the parent node and all the other nodes in the graph must be children of the scene node. In Figure 2, "dcs" represents dynamic coordinate system, and "pf" indicates a Performer™ feature.
In VADE, when the part is in the bin it is a child of the global coordinate system. Once the part is grasped, it becomes a child of the palm, which is a child of the hand (as seen in Figure 2). On the other hand the base part does not come under any such hierarchy and can be manipulated using the left hand. When the grabbed part is assembled to the base part, its DCS becomes a child of the base part.

**Constrained Motion Methodology**

This section mentions a few salient points about the methods presented in this paper, and some other concepts to be kept in mind to serve as a link to the following sections.

When being manipulated in the virtual environment, the object grabbed by the right hand is checked to see if it is close enough to be constrained with the base part of the assembly. As mentioned earlier, most constraints in real-world assemblies involve axial alignment and/or surface mating (with or without offsets). Along with the constraint information, the final location and orientation of each component in the assembly is extracted from the CAD database. This is stored as a 4 x 4 transformation matrix. For each constraint of the grabbed part, the above transformation matrix for the part is used to obtain the corresponding constraint on the base part.
During assembly, each constraint of the grabbed part and the corresponding constraint on the base part are checked to identify if they are close enough to be constrained. If certain conditions are satisfied, then the object is snapped on to a constrained space and restricted to move only in certain allowable directions. For example, if a shaft is being assembled into a hole in the base part and the axis on the shaft and the axis of the hole are close enough to be aligned, then the conditions for axial constraints are met, and the shaft is constrained to move only along this common axis or rotate about the axis.

In Pro/ENGINEER, which is the CAD system used for this work, the assembly operations are comparable with the operations of the actual assembly process in the real world. Again taking the example of assembling a nut on a bolt, the axis of the nut and the axis of the bolt can be aligned. Next, the plane of the nut and plane of the bolt-head can be mated using the mate surfaces option. Pro/ENGINEER also provides ‘Mate Offset’ and ‘Align Offset’ options to allow two constrained surfaces to be offset. In addition, it provides various other constraint options including coordinate system constraints, constraining surfaces to be parallel, and constraining male and female surfaces to mate.

The nature of information obtained from Pro/ENGINEER for implementing the constraint methods are as follows:

For axis constraints, two points in space defining the ends of the graphical line representing the axis are obtained. In addition, the type of constraint (align or mate), the offset, (if any), between the two axis under consideration are obtained. For plane constraints, three unit vectors and the origin defining the plane are obtained. One of the unit vectors is the normal vector of that plane, starting at the origin of the plane. The type and offset are also obtained in this case.

The next four sections explain in-depth the methods for axial and planar constraints:

- Checking for Axis Constraints - the two axis under consideration are checked to see if they are close enough to be constrained.
- Application of Axis Constraints - the axial constraint is applied to the motions of the grabbed part allowing only translations along the axis and rotations about the axis.
- Checking for Plane Constraints - the two planes under consideration are checked to see if they are close enough to be constrained.
• Application of Plane Constraints - the planar constraint is applied to the motions of the grabbed part allowing movements only along the plane and rotations about the normal vector of the plane.

The checking and application of all the constraints for a given component are performed during every frame.

Some guidelines and naming conventions used to denote points, vectors, and matrices in the following sections are as follows:

1. — on top of the text denotes a point in 3D space, 3 (x, y, z) values.
2. → on top denotes a vector.
3. ∧ on top denotes a unit vector (normalized).
4. [ ] denotes a matrix. The matrices which are frequently used and the matrices which are used to obtain the final transformation matrix for axis and plane constraints are denoted in bold letters.
5. Subscript ‘g’ means the vector or the point is in the global coordinate system. Similarly subscripts ‘p’ & ‘b’ denote the grabbed part coordinate system and base part coordinate system respectively.
6. Any reference to ‘part’ in the following sections refers to the grabbed part.
7. A ‘.’ denotes dot product, ‘x’ denotes multiplication, and x (in bold) denotes cross product.

The transformation matrices [part_matrix], [palm_matrix], [hand_matrix], and [baseLocationXform] are required to transform each child DCS to its final location and orientation with respect to its parent DCS (Figure 3).

• [part_matrix] (P1) transforms an object from the part’s coordinate system to the palm’s coordinate system.

• [palm_matrix] (P2) transforms an object from the palm’s coordinate system to the hand’s coordinate system

• [hand_matrix] (H) transforms an object from the hand coordinate system to the global coordinate system as seen in the Figure 3.
Also \([\text{partLocationXform}] (pLX)\) is the transformation matrix used to take a point from the part's local coordinate system (CS) to the global CS. It is obtained as shown in the equation below:

\[
[\text{partLocationXform}] = [\text{part_matrix}] \times [\text{palm_matrix}] \times [\text{hand_matrix}]
\] (1a)

or, \([pLX] = [P1] \times [P2] \times [H]\) (1b)

Next the transformation to take a point from the base part's CS to the grabbed part's CS is obtained by the following equations:

\[
[\text{invPartLocationXform}] = [\text{partLocationXform}]^{-1}
\] (2)

\[
[\text{baseInPartLocationXform}] = [\text{baseLocationXform}] \times [\text{invPartLocationXform}]
\] (3)

where \([\text{baseLocationXform}]\) is the transformation matrix which locates the base part in the global CS. The transformation matrices obtained in equations 1, 2, & 3 will be used frequently in the following sections.

![Diagram](image)

**Fig. 3. Transformations from the part CS to the global CS.**

**Checking for Axis Constraints**

The overall philosophy for checking axis constraints is as follows: 1) the axis on the grabbed part and the axis on the base part are compared in the same coordinate system, 2) the perpendicular distance from a point on the part's axis to the base part axis is calculated, 3) the two axis are said to be close enough to be constrained if certain conditions are met, and 4) if they are close enough to be constrained, then the axial constraint is applied to the motion of the grabbed part. This section explains the conditions which need to be satisfied before the axial constraint is applied to the part.
To check for axis constraints, the axis on the part and the axis on the base part are compared in global coordinates. The two points on each axis need to be transformed to the global CS. Using [partLocationXform] obtained in equation 1, the endpoints of the axis on the part are transformed to global coordinates. Similarly, [baseLocationXform] is used to transform the endpoints of the axis on the base part to the global CS. The equations to obtain the four points in the global CS (Figure 4) are given below:

\[
(\overrightarrow{ep_1})_g = (\overrightarrow{ep_1})_p \times [pLX]
\]  \hspace{1cm} (4)

\[
(\overrightarrow{ep_2})_g = (\overrightarrow{ep_2})_p \times [pLX]
\]  \hspace{1cm} (5)

\[
(\overrightarrow{eb_1})_g = (\overrightarrow{eb_1})_b \times [baseLocationXform]
\]  \hspace{1cm} (6)

\[
(\overrightarrow{eb_2})_g = (\overrightarrow{eb_2})_b \times [baseLocationXform]
\]  \hspace{1cm} (7)

Next the vectors $\overrightarrow{Vec1}$, $\overrightarrow{Vec2}$ and $\overrightarrow{Vec3}$ (shown in Figure 4) are obtained as shown below:

\[
\overrightarrow{Vec1} = (\overrightarrow{eb_2})_g - (\overrightarrow{eb_1})_g
\]  \hspace{1cm} (8)

\[
\overrightarrow{Vec2} = (\overrightarrow{ep_1})_g - (\overrightarrow{eb_1})_g
\]  \hspace{1cm} (9)

\[
\overrightarrow{Vec3} = (\overrightarrow{ep_2})_g - (\overrightarrow{ep_1})_g
\]  \hspace{1cm} (10)

The three vectors are then normalized. The two conditions that are checked are the dot product of the two axis and the perpendicular distance between the point $(\overrightarrow{ep_1})_g$ and $\overrightarrow{Vec1}$. If the two axis are close enough to be parallel, then the dot product of $\overrightarrow{Vec1}$ and $\overrightarrow{Vec3}$ must be very close to 1.0 (Figure 4). A tolerance is necessary to compensate for the inherent 'inaccuracies' of virtual reality based manipulation. Thus, the dot product is checked against an axis_tolerance value.

\[
\left(\overrightarrow{Vec1}\right) \cdot \left(\overrightarrow{Vec3}\right) \geq \text{axis}\_\text{tolerance}
\]  \hspace{1cm} (11)
If equation 11 is satisfied, the first condition is met. The cross product of $\hat{\text{Vec}}2$ and $\hat{\text{Vec}}1$ gives a vector $\hat{\text{Vec}}4$ in the direction perpendicular to the plane determined by the two vectors $\hat{\text{Vec}}2$ and $\hat{\text{Vec}}1$. $\hat{\text{Vec}}4$ is then normalized to give $\hat{\text{Vec}}4$. The cross product of $\hat{\text{Vec}}4$ and $\hat{\text{Vec}}1$ gives a vector at right angles to $\hat{\text{Vec}}1$ and on the plane defined by $\hat{\text{Vec}}2$ and $\hat{\text{Vec}}1$. This is denoted by $\hat{\text{Vec}}5$ in Figure 4 and is given by the following equations.

$$\vec{\text{Vec}}4 = \hat{\text{Vec}}2 \times \hat{\text{Vec}}1$$  \hspace{1cm} (12) \\
$$\hat{\text{Vec}}5 = \hat{\text{Vec}}4 \times \hat{\text{Vec}}1$$  \hspace{1cm} (13)

![Diagram showing the relationship between vectors and coordinate system](image)

**Fig. 4. Checking for axis constraints.**

The operation in equation 13 may be thought of as merely rotating $\hat{\text{Vec}}1$ by 90 degrees on the plane defined by $\hat{\text{Vec}}1$ and $\hat{\text{Vec}}2$. The dot product of unit vector $\hat{\text{Vec}}5$ and $\hat{\text{Vec}}2$ will give the required perpendicular distance (D1) as shown below.

$$D1 = \hat{\text{Vec}}5 \cdot \hat{\text{Vec}}2$$  \hspace{1cm} (14)
The absolute value of distance \( D1 \) is checked against a tolerance value for the same reason mentioned earlier.

\[ |D1| \leq D1\_tolerance \] (15)

If equation 15 is satisfied, then the second condition is met and axial constraint is applied to the motion of the part.

**Application of Axis Constraints**

After checking for axis constraints, the next step is to apply the constraint to the motion of the grabbed part. The application of axis constraints is a little more complicated than checking for the axis constraints. There are three main steps involved and each of these is explained separately.

1) Move the part so that the axis on the part is at the origin of the part CS. This step can be neglected if axis on the part passes through the origin.

2) Rotate the axis on the part so that it is parallel to the axis on the base part.

3) Move the part so that the two axes are aligned.

The transformation matrices obtained from each of these are multiplied together to get the final transformation matrix.

The first step is to obtain a translation matrix which moves the axis on the part to the origin of the part CS. Since the calculations are being done in the part’s CS, the endpoints of the axis on the part do not need to be transformed. The difference between the two endpoints is taken and normalized to obtain unit_part_vector \( \left( \frac{\wedge{u pv}}{p} \right) \) (Figure 5a).

\[
\left( \frac{\wedge{u pv}}{p} \right) = \left( \frac{ep_2}{p} \right) - \left( \frac{ep_1}{p} \right)
\] (16)

The dot product of unit_part_vector \( \left( \frac{\wedge{u pv}}{p} \right) \) and negative of the vector \( \left( \frac{ep_1}{p} \right) \) (obtained by subtracting the origin from the point \( ep_1 \)) gives the magnitude of \( \left( \frac{ep_1}{p} \right) \) in the direction of the
unit_part_vector. This magnitude when multiplied by the unit_part_vector \((\hat{upv})_p\) gives the testing_vector denoted by \(\hat{T}\) in the Figure 5a. It is so called because it can be used to check if this step in the application of axis constraints can be eliminated. If the axis on the part passes through the origin, then this testing_vector \((\hat{T})\) will be the same as \((-ep_1)_p\) and this first step can be neglected.

\[
\hat{T} = \left(\left(\left(-ep_1\right)_p \cdot (\hat{upv})_p\right) \times (\hat{upv})_p\right)
\]

(17)

Using testing_vector \(\hat{T}\) and vector \(ep_1\), vector \(sv\) is obtained.

\[
sv = \hat{T} + \left(ep_1\right)_p
\]

(18)

**Fig. 5a. Application of axis constraints.**

The vector obtained in equation 18 translates the origin of the part to the axis. Thus \(sv\) is negated and the components of the vector are put in matrix form to get the translation matrix [sv_NegXform], as shown below:

\[
sv_{NegXform} = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
(-sv) \times (-sv) \times (-sv)
\end{bmatrix}
\]

(19)
The translation matrix $[\text{sv\_NegXform}]$ moves the axis so that it passes through the origin of the part CS and will be used in the final step.

The next step is to obtain the transformation matrix which rotates the axis on the part so that it is parallel to the axis on the base part. In equation 11, a tolerance is used to test if the two axes are “almost” parallel. Hence a small rotation needs to be applied to the part to make the two axes “exactly” parallel. This step is required for the accuracy of the data being generated by the virtual assembly system.

Since this calculation is also performed in the part’s coordinate system, the endpoints of the axis on the base part are transformed to the part’s CS using $[\text{baseInPartLocationXform}]$ (from equation 3), as given below:

$$\left(\overline{eb_1}\right)_p = \left(\overline{eb_1}\right)_b \times [\text{baseInPartLocationXform}] \quad (20)$$

$$\left(\overline{eb_2}\right)_p = \left(\overline{eb_2}\right)_b \times [\text{baseInPartLocationXform}] \quad (21)$$

The difference between the two endpoints ($\overline{eb_1}$ and $\overline{eb_2}$) is normalized to obtain the unit base vector $\left(\hat{ubv}\right)_p$.

$$\left(\hat{ubv}\right)_p = \frac{\left(\overline{eb_2}\right)_p - \left(\overline{eb_1}\right)_p}{6 4 4 7 4 4 8} \quad (22)$$

Figure 5b shows $\left(\hat{ubv}\right)_p$ (from above) and $\left(\hat{upv}\right)_p$ from (equation 16). A transformation matrix which rotates unit part vector $\left(\hat{upv}\right)_p$ on to unit base vector $\left(\hat{ubv}\right)_p$ is obtained and is called $[\text{axisRotate}]$. 
Fig. 5b. Application of axis constraints.

The last step is to move and snap the part’s axis on to the base part’s axis so that they are aligned. Since the previous two steps move the part’s constraint axis to the origin of the part CS and rotate the part to the correct angle in this step, the part needs to be moved so that its origin lies on the base part’s constraint axis. This calculation is performed in the global CS. Equations 6 & 7 are used to obtain the endpoints of the base part’s axis in global coordinates. Equation 1b gives \( [pLX] \) which is the matrix transformation from the part’s CS to the global CS. The translational component of this matrix is denoted by vector \( \vec{P} \) in Figure 5c.

\[
\vec{P} = (pLX_x, pLX_y, pLX_z)
\]  

(23)

In global CS, the vector from origin to \( (eb_1)_g \) is denoted by \( \vec{d} \), and is used to obtain vector \( \vec{b} \) (Figure 5c).

\[
\vec{b} = \vec{P} - \vec{d}
\]  

(24)
Further, vector $\hat{\rho}$, and points $\left(\hat{e}_b \right)_g$ and $\left(\hat{e}_b \right)_g$ are used to calculate vector $\vec{\rho}$ as given by the following equations:

$$\begin{bmatrix} \hat{u}v \end{bmatrix}_g = \left(\hat{e}_b \right)_g - \left(\hat{e}_b \right)_g$$  \hspace{1cm} (25)

$$\vec{\rho} = \left( \hat{B} \cdot \begin{bmatrix} \hat{u}v \end{bmatrix}_g \right) \times \left( \begin{bmatrix} \hat{u}v \end{bmatrix}_g \right)$$  \hspace{1cm} (26)

The $\vec{\text{partLocation}}$ vector is obtained from vectors $\vec{\rho}$ and $\vec{\hat{\rho}}$.

$$\vec{\text{partLocation}} = \vec{\rho} - \vec{\hat{\rho}}$$  \hspace{1cm} (27)

Now $\vec{\text{partLocation}}$ needs to be transformed to the part's CS and since it is a vector and not a point, the translational components of $[\text{invPartLocationXform}]$ are set to zero before transformation.

$$\vec{fPL} = \vec{\text{partLocation}} \times [\text{invPartLocationXform}] \text{(with no translation)}$$  \hspace{1cm} (28)

Finally $\vec{fPL}$ is put in matrix form $[\text{partTranslationXform}]$ so that it can be used with other matrices obtained from the first 2 steps.
\[
\text{partTranslationXform} = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
\text{fPL}_x & \text{fPL}_y & \text{fPL}_z & 1
\end{bmatrix}
\] (29)

The transformation matrices obtained in equations 19, 29, and [axisRotate] are multiplied to get [absFinalPartXform]_{axis}.

\[
[\text{absFinalPartXform}]_{axis} = [\text{sv\_NegXform}] \times [\text{axisRotate}] \times [\text{partTranslationXform}]
\] (30)

[sv\_NegXform] takes the part’s axis to the origin of the part CS. Next [axisRotate] makes sure that the two axis under consideration are parallel. [partTranslationXform] snaps the part’s origin on to the base part’s axis so that the two axes are aligned and constrained. When these axial constraints are applied, the only allowable motions are translation along that axis and rotation about the axis.

**Checking for Plane Constraints**

The checking of plane constraints is partly similar to checking axis constraints. The normal vectors of the two planes are considered as axes which need to be parallel. The overall philosophy for checking plane constraints is as follows: 1) the normal vector of the plane on the grabbed part and the normal vector of the plane on the base part are compared in the same coordinate system to identify if they are almost parallel, 2) the distance between the two planes under consideration is calculated, 3) the two planes are said to be close enough to be constrained if certain conditions are met, and 4) if they are close enough to be constrained, then the plane constraint is applied to the motion of the grabbed part. This section explains the conditions which need to be satisfied before the application of the plane constraint.
In planar constraints the data we are interested in are the normal vector of the plane which is a unit vector and the origin of the plane which is a point. It needs to be kept in mind that, when transforming the normal of the plane, the transformation matrix used should have its translational components set to zero, since we are transforming a vector and not a point. When transforming the origin of the plane, no change needs to be made to the transformation matrices.

To check for planar constraints, the plane on the part and the plane on the base part are compared in global coordinates. First the origin on the part plane and base part plane are transformed to the global CS as shown below:

\[
\begin{align*}
\overrightarrow{(p\_origin)}_g &= \overrightarrow{(p\_origin)}_p \times [partLocationXform] \\
\overrightarrow{(bp\_origin)}_g &= \overrightarrow{(bp\_origin)}_b \times [baseLocationXform]
\end{align*}
\] (31) (32)

Next the translational components of the two transformation matrices used above are set to zero to obtain \([partLocationXform\_vector]\) and \([baseLocationXform\_vector]\). These are then used to transform the normals on both the planes to the global CS.

\[
\text{partLocationXform\_vector} = \begin{bmatrix}
\text{unchanged}_x \\
\text{unchanged}_y \\
\text{unchanged}_z \\
0 \\
0 \\
0 \\
0 \\
0 \\
1
\end{bmatrix}
\] (33)

\[
\text{baseLocationXform\_vector} = \begin{bmatrix}
\text{unchanged}_x \\
\text{unchanged}_y \\
\text{unchanged}_z \\
0 \\
0 \\
0 \\
0 \\
0 \\
1
\end{bmatrix}
\] (34)

\[
\left(\overrightarrow{p\_normal}\right)_g = \left(\overrightarrow{p\_normal}\right)_p \times [partLocationXform\_vector]
\] (35)

\[
\left(\overrightarrow{bp\_normal}\right)_g = \left(\overrightarrow{bp\_normal}\right)_b \times [baseLocationXform\_vector]
\] (36)

The two conditions that are checked are the dot product of the two normals and the distance between the planes. If the dot product of the normal on the part’s plane and the normal on the base part’s plane is close to 1.0, then the two planes are almost parallel. Thus the dot product is checked against a normal_tolerance value.
\[ (bp_{\text{normal}})^g \cdot (p_{\text{normal}})^g \geq \text{normal\_tolerance} \]  

(37)

If equation 37 is satisfied, the first condition is met. Next, the perpendicular distance between the planes is calculated. The difference between the origin on the grabbed part’s plane and the origin on the base part’s plane gives a vector called \( \text{diff\_origins}^g \) as shown in the Figure 6.

The dot product of this vector \& the normal on the base part’s plane gives the distance between the planes as given by the following equations:

\[ \text{distance} = (\text{diff\_origins}^g)^g \cdot (bp_{\text{normal}})^g \]  

(38)

The distance obtained from equation 39 is compared against a distance\_tolerance. This tolerance is necessary because it is not possible to exactly mate planes in virtual environments due to the limitations of the tracking devices, and human motor skills.

Fig. 6. Checking for plane constraints.

\[ \text{distance} \leq \text{distance\_tolerance} \]  

(40)

If equation 40 is satisfied, then the second condition is met and the plane constraint is applied to the motion of the part.
Application of Plane Constraints

After checking for plane constraints the next step is to apply the constraint to the motion of the grabbed part. The application of plane constraints can be explained relatively easily since the axial constraints concepts have already been established. There are four steps, and matrices obtained from each of these are multiplied to get the final transformation matrix.

1) Move the origin of the part’s constraint plane to the origin of the part CS.
2) Rotate the normal on the part plane so that it is exactly parallel to the normal on the plane of the base part.
3) Move the origin of the part plane back to its original position.
4) Move the part plane so that it aligns with the plane on the base part.

All the calculations that follow are performed in the part’s CS, hence the origin and normal of the plane on the base part are transformed to the part’s CS.

$$\left( \overrightarrow{bp\_origin} \right)_p = \left( \overrightarrow{bp\_origin} \right)_b \times [baseInPartLocationXform]$$  \hspace{2cm} (41)

The translational components of $[baseInPartLocationXform]$ are set to zero to obtain $[baseInPartLocationXform\_vector]$, which is used to transform the normal to the part’s CS.

$$baseInPartLocationXform\_vector = \begin{bmatrix} \text{unchanged} \\ - \\ - \\ 0 0 0 1 \end{bmatrix}$$  \hspace{2cm} (42)

$$\left( \overrightarrow{bp\_normal} \right)_p = \left( \overrightarrow{bp\_normal} \right)_b \times [baseInPartLocationXform\_vector]$$  \hspace{2cm} (43)

The vectors obtained from equations 41 & 43 are shown in Figure 7.

The first step is to obtain a translation matrix which moves the origin of the part plane to the origin of the part CS. But in this case, we already have the origin of the part’s plane in the part CS. The components of this origin are put in matrix form to get $[p\_originXform]$. 
\[
p_{\text{originXform}} = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
(p_{\text{origin}})_{p_x} & (p_{\text{origin}})_{p_y} & (p_{\text{origin}})_{p_z} & 1 \\
\end{bmatrix}
\]

(44)

The location of the origin of the plane is negated and its components are stored in a matrix called \([p_{\text{originNegXform}}]\).

\[
p_{\text{originNegXform}} = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
(-p_{\text{origin}})_{p_x} & (-p_{\text{origin}})_{p_y} & (-p_{\text{origin}})_{p_z} & 1 \\
\end{bmatrix}
\]

(45)

The two matrices \([p_{\text{originNegXform}}]\) and \([p_{\text{originXform}}]\) will be used in the final step.

The next step is to calculate a transformation matrix which rotates the normal on the part’s plane so that it is parallel to the normal on the base part’s plane. In equation 37, a tolerance is used to test if the two normals are “almost” parallel. Hence a small rotation needs to be applied to make the two normals “exactly” parallel. A transformation matrix which rotates \((p_{normal})_p\) on to \((bp_{normal})_p\) is obtained. The resultant matrix, \([\text{normalRotate}]\) is used in the end to calculate the final transformation matrix.

The last step is to move part’s plane by the required distance in the required direction and snap the two planes together. For this, the distance to move the plane along the normal on the base part’s plane needs to be calculated. Referring to Figure 7, if the negative of the distance between the planes is scaled by the normal vector of the base part’s plane, \(distance_{bp_{normal}}\) is obtained as shown below:

\[
\left(\text{diff}_\text{origins}\right)_p = (p_{\text{origin}})_p - (bp_{\text{origin}})_p
\]

(46)

\[
distance_{bp_{normal}} = \overrightarrow{dbn} = \left(\text{diff}_\text{origins}\right)_p \cdot (bp_{normal})_p \times (bp_{\text{normal}})_p
\]

(47)

The components of the vector obtained from the above equation are put in matrix form to get \([distance_{bp_{normalXform}}]\).
\[
\text{distance\_bp\_normalXform} = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
\text{dbn}_x & \text{dbn}_y & \text{dbn}_z & 1
\end{bmatrix}
\] (48)

Fig. 7. Application of plane constraints.

The transformation matrices from equations 44, 45, 48, & [normalRotate] are multiplied to get \([\text{absFinalPartXform}]_{\text{plane}}\).

\[
[\text{absFinalPartXform}]_{\text{plane}} = [\text{p\_originNegXform}] \times [\text{normalRotate}] \times [\text{p\_originXform}] \times [\text{distance\_bp\_normalXform}]
\] (49)

[\text{p\_originNegXform}] moves the origin of the plane on the part to the origin of the part. Next, [normalRotate] makes sure the two planes are exactly parallel. Then [\text{p\_originXform}] takes the origin of the part plane back to its original position. Finally [\text{distance\_bp\_normalXform}] snaps and constrains the two planes together by moving the part plane in the required direction.

**Implementation Details**

**Hardware:**

The primary workstation used for this research was a Crimson™ workstation by Silicon Graphics, Inc. It has a single, 150mhz, MIPS R4400 processor, with 128 MB of RAM, Reality
Engine™ Graphics, and a multi-channel option board. Global position and orientation tracking was done by the Ascension Flock of Birds™ system with an Extended Range Transmitter (ERT). This transmitter employs a pulsed, DC magnetic field and is capable of determining 6 DOF information from each of its receivers. Three receivers are used in this system, one to track the head so that the user can ‘look around’, another to track the right hand and the last one is held in the left hand (for moving the base part) facilitating assembly operations.

The CyberGlove™ by Virtual Technologies Inc. is used to monitor the finger and wrist movements. This 22 sensor glove served to augment the graphical representation of the right hand. It measures the bending and abduction motions of the wrist joint, the bending of the three joints on all four fingers and the thumb, the abduction between all four fingers, the arch of the palm, and the abduction of the thumb from the palm. The digitized output values from the sensors of the glove are converted to appropriate joint angles for a specific user’s hand using a calibration function. These joint angles are compared against a glove tolerance to facilitate releasing the part when the user stretches his/her hand to drop the part. The VR4 immersive head-mounted display is used for graphical display.

**Software:**

The graphical basis for the environment created is the Silicon Graphics IRIS Performer™ Library. IRIS Performer™ is a software toolkit for the development of real-time 3D graphics, visualization, and simulation applications. Performer™ sits “on top” of Silicon Graphics OpenGL™ libraries. It also has better optimization of its own functions and in turn allowed better performance when using complex models.

Due to its wide-spread popularity and functionality as a designing tool, Pro/ENGINEER™ was chosen as the software for the creation of the CAD models for use in this system. Pro/DEVELOP™ is the developer’s toolkit for Pro/ENGINEER™, and is designed to be used as a means to access the Pro/ENGINEER™ database. The Pro/DEVELOP™ module was used to automate data exchange between the CAD system and the virtual assembly system.

It needs to be noted that information from only the first level within the assembly tree is transferred to the virtual assembler (Figure 8). This was done because any sub-assembly that is a
part of an assembly can be assembled using the system in a separate session. Also the graphical representations of the part models have to be supplied to the assembler. The format chosen was the Inventor™ file format developed by Silicon Graphics for use with the OpenInventor™ graphics library. The data files are manually created by the designer, in the Pro/ENGINEER™ environment, for each of the level-one parts and sub-assemblies.

![Diagram of an assembly tree](image)

**Fig. 8. Example of an assembly tree.**

During the assembly process in the virtual environment, once the motions of the part are constrained, even if the base part is translated or rotated, the movements of the grabbed part still remain constrained. It is possible to release a grasped part and re-grab it in a different orientation if necessary. Since gravity is not yet incorporated within the system, the part floats in mid-air when released. When released, the constraints that were applied on the part are removed.

**Test Cases**

Several test cases were conducted to demonstrate the importance of incorporating constraints in virtual environments. Two assemblies were chosen. The first one was a simple assembly chosen to illustrate the concepts clearly. The second assembly was a more complex one to show that this assembly planning system can be used for realistic industrial applications. In this section only the simpler assembly testing is presented, illustrating the usefulness of constrained motion methodology.
Figure 9 shows a ‘white base part’ placed on the table and a ‘red block’ which has 2 holes which align with the two corresponding holes on the base part when assembled. This assembly operation was performed in two ways: 1) by taking the block close to the top face of the base part, applying the planar constraints and moving it to its final location and orientation, and 2) by doing the same without the help of constraints. Figures 10 & 11 show the results for these two cases. In both these figures, the path of the grabbed part is shown in the form of a crude swept volume. As can be seen in Figure 10, the planar constraints have been applied for the block, so the bottom plane of the block is constrained to move along the top plane of the base part. With the plane constraints applied, the block slides smoothly on the plane of the base part. For the case where constraints are not used, the block cuts through the base part as is seen in Figure 11. Because of the shakiness of the human hand and inaccuracies of hardware devices, for the case where constraints are not applied, the assembly operation does not simulate the physical reality of constraints. If the swept volume information is to be sent back to the CAD system, it will contain unnecessary and undesired information if constraints are not used.

The next operation is to assemble a pin into the hole in the block. Figure 12 shows the gripped blue pin ready for assembly. Figures 13 & 14 show this operation with and without the application of axis constraints respectively. Again, a crude swept volume is used in these two figures to track the movement of the pin. As seen in Figure 13, the path of the pin is not affected by the jitters of the hand, since the only allowed movements are along and about that axis. When axis constraints are not applied, it is evident from Figure 14 that though the user is trying to move the pin only along the axis, the path followed is not along the desired direction. When evaluating tolerance issues, such differences and lack of precision are not acceptable.
Fig. 9. Block being assembled on plate.

Fig. 10. Block sliding on plate with planar constraint.
Fig. 11. Block sliding on plate without planar constraint.

Fig. 12. Pin being assembled to block.
Fig. 13. Pin assembly using axial constraint.

Fig. 14. Pin assembly without axial constraint.
Conclusions

This research addressed certain key issues involved in “design by manufacture”. This virtual assembly planning system offers designers a unique tool for achieving the goals of a useable, manufacturable, and assemblable product. The significance of incorporating constrained motions in virtual environments has been elucidated. The test cases clearly show the importance of applying constrained motions to evaluating tolerance issues and sending softzone data to the CAD system.

The mathematical methods presented in this paper allow assembly engineers to use virtual reality systems to manipulate and assemble parts while simulating the physical reality of contact constraints. Thus, the data generated for CAD analysis is numerically accurate and the overall environment is also realistic.

Although the implementation of constrained part motion within the virtual assembly system gives the designer an illusion of solidity in the models being used, true collision detection would only enhance the quality of the experience. The movement of one solid part through another is not a valid method of assembly, although it can provide insight to improve the design. Continuation of this work involves constraint management methods, complex surface constraints, enhanced gripping capability, physics-based modeling, and complete softzone generation capability.

Acknowledgments

The authors acknowledge the support provided by the National Institute for Standards and Technology (NIST), Manufacturing Systems Integration Division, through grant number 60NANB5D0066.
References


