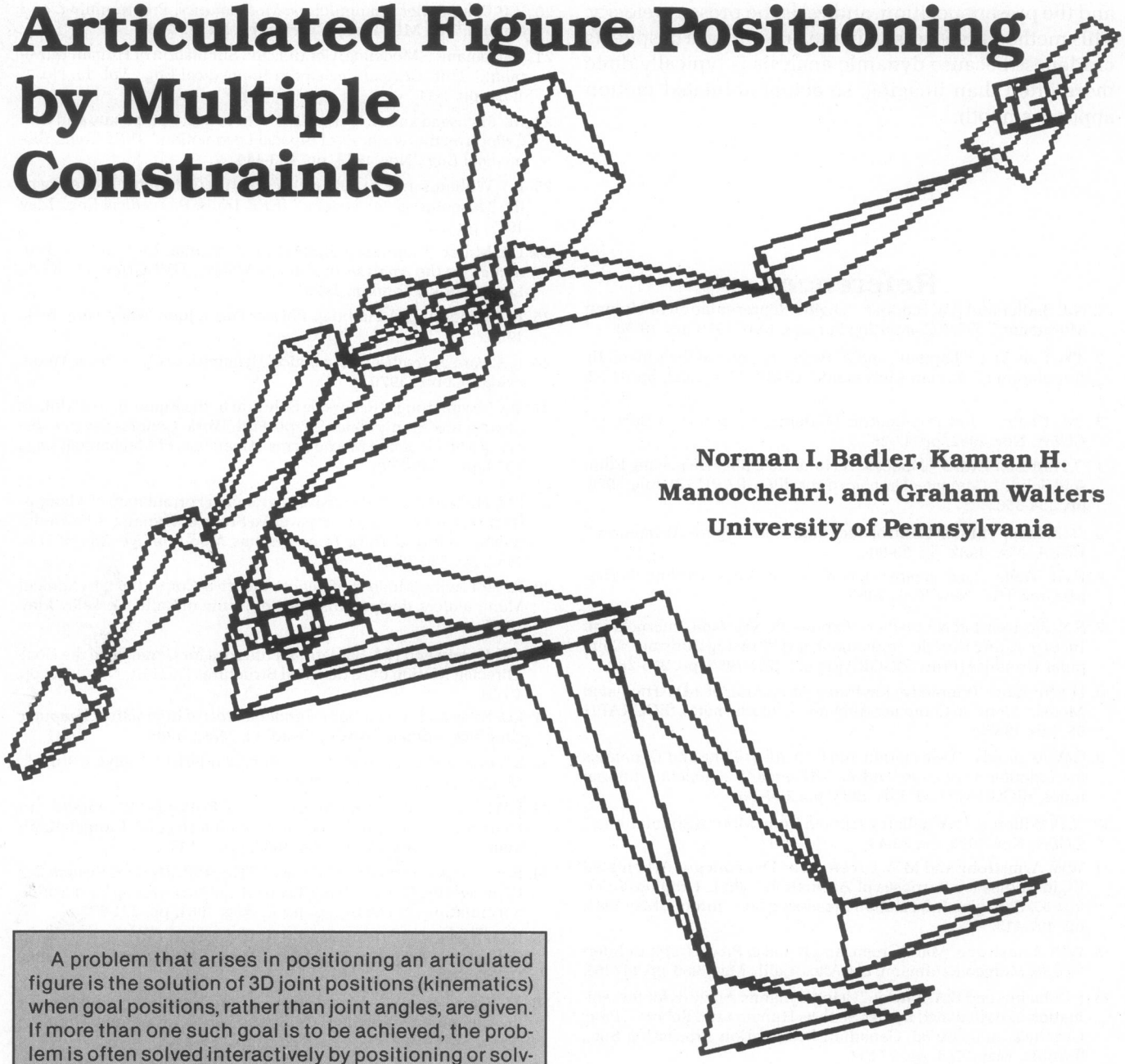
Articulated Figure Positioning by Multiple Constraints



Norman I. Badler, Kamran H. Manoochehri, and Graham Walters
University of Pennsylvania

A problem that arises in positioning an articulated figure is the solution of 3D joint positions (kinematics) when goal positions, rather than joint angles, are given. If more than one such goal is to be achieved, the problem is often solved interactively by positioning or solving one component of the linkage, then adjusting another, then redoing the first, and so on. This iterative process is slow and tedious. We present a method that automatically solves multiple simultaneous joint position goals. The user interface offers a six-degree-of-freedom input device to specify joint angles and goal positions interactively. Examples are used to demonstrate the power and efficiency of this method for key-position animation.

Key positioning is an animation technique where objects are placed in different positions and an interpolation method[1-3] computes the in-between frame-to-frame parameter values. When articulated figures are animated, the joints undergo rotations in addition to the translation and rotation of the whole body. Interpolating the joint angles makes the figure wave, gesture, walk, or dance. The key-positioning task itself will be addressed here, rather than the motion-control issues, which are adequately addressed in the other articles in this issue of *CG&A*.

The positioning task has always been difficult for animators, and the difficulty is even more apparent with a highly articulated figure such as a human body. For example, the model of the human body used by TEMPUS[4] has 18 joints and 48 degrees of freedom. Use of input devices such as a mouse (with two degrees of freedom), dials, or a keyboard may be awkward since many joints of the body have more degrees of freedom

0272-1716/87/0600-0028$01.00 © 1987 IEEE

than the input devices. Using a multiplicity of devices presents a user-control problem. Moreover, there is little kinesthetic correspondence between the orientation of a limb segment and the physical structure of a mouse, dial, etc.

There have been many efforts to connect an actual body physically to joint angle sensors and use this information to describe joint parameters for an animation.[5,6] The difficulties with this approach are the delicacy and awkwardness of the sensor setup and, more importantly, the specificity of the resulting position or motion to the user's or performer's body size. Much work has been done to facilitate the joint-positioning task and improve its generality through algorithmic assistance.

Positioning can be done manually, meaning the user has to specify the angle of each joint of the figure one at a time, or it can be done with some kinematic assistance. *Inverse kinematics* finds, given an arbitrary chain of joints and a position in space, the joint angles such that the distal end of the chain reaches that position in space. Solving this problem has been difficult when dealing with articulated figures, especially when redundant degrees of freedom are present (as in a human figure). One solution is the inverse or pseudo-inverse Jacobian matrix, used in the field of robotics[7] and lately used in the control of animation of legged figures (see the article by Girard in this issue of *CG&A*).[8,9] Another solution is described by Korein,[10] who defines the movement limits of each joint as a spherical polygon: The intersection of the polyhedra created by sweeping the segments through the spherical polygon joint limits leads to a recursive formulation of the reachable workspace of the limb.

Positioning by constraints

Constraints or goals may also be used to position a figure. By "constraint" we mean a desired relationship between entities, but the method of achievement is not known a priori. Constraints themselves are not new to computer graphics; they were used as early as the 1960's in Sutherland's Sketchpad.[11] Their application to 3D articulated figure key positioning, however, has not been exploited. O'Rourke and Badler,[12] and Marion, Fleischer, and Vickers[13] have done some studies using constraints for positioning, though only the former have worked with a complete 3D figure. Also, Coblentz, Gueneau, and Bonjour[14] have suggested a method for finding the optimal posture in a sitting position. This method, however, is limited to a 2D figure with eight degrees of freedom. Both Korein and Girard have applied end-effector position constraints to permit limited modification of the available degrees of freedom between the fixed proximal and distal ends of a chain.[8,10] Our method of positioning is a mixture of multiple constraints and inverse kinematics.

Positioning a figure often involves several simultaneous and even approximate constraints. For example, animating a body in the process of sitting in a chair involves multiple constraints between the subject's body segments and joints with the back, legs, and seat of the chair. We want the body's hips to be located nearly on the chair, its thighs to lie parallel to the seat, and the center of its back to lie against the back of the chair, but the exact positions of these respective goals are known only approximately. Since in solving for a single reach we have to know the position of the goals exactly, formulating the "sitting" process as a series of single reaches would be very difficult. The achievement of one goal is apt to cause other previously satisfactory joint positions to be displaced. The only recourse would be to try various reaches repeatedly, a task further complicated if the display is slow or the image not easily manipulated.

To avoid the inefficiency of creating one goal at a time, we instead create many goals for different points on the figure and try to solve for the best position that satisfies these goals. Each goal has a weight, which is interpreted as its "importance": If it is not possible for each point to reach its goal, the algorithm uses the weight values to decide which points must be closer to the goal and which can be permitted to reside farther away. Weights are assigned by the user, based on the relative desirability or enforcement of a constraint; the higher weighted goals will be achieved at the expense of the lower weighted ones.

We can imagine that for each segment goal there exists a spring connecting the distal joint of the segment and the goal, and each such spring has a (possibly) different spring constant (its weight value). To let these springs act on the body and change its joint angles and position in space is to solve for the desired position. Since the connectivity and segment lengths of the figure must remain constant, the main consequence of this approach is that the changes induced by the spring forces must be reflected by changing joint angles or the whole body position.

We designed and implemented a system called POSIT to investigate articulated figure positioning by the method of multiple constraints.[15] POSIT is implemented on a Silicon Graphics Iris 3030 workstation, for real-time visual feedback. For interactive input, two devices are used: a mouse and a six-degree-of-freedom sensor—a 3SPACE digitizer system (from Polhemus Navigation Sciences Division, McDonnell Douglas Electronics Company), hereafter simply called the Polhemus. The Polhemus senses six degrees of freedom within a cubic meter region of space at a sample rate of approximately 40 Hz. The principal benefit of the Polhemus is that it permits the user to interact in 3D space in a kinesthetically reasonable fashion.[16,17]

A constraint-satisfaction algorithm is used to enhance achievement of a desired configuration. Each joint of the

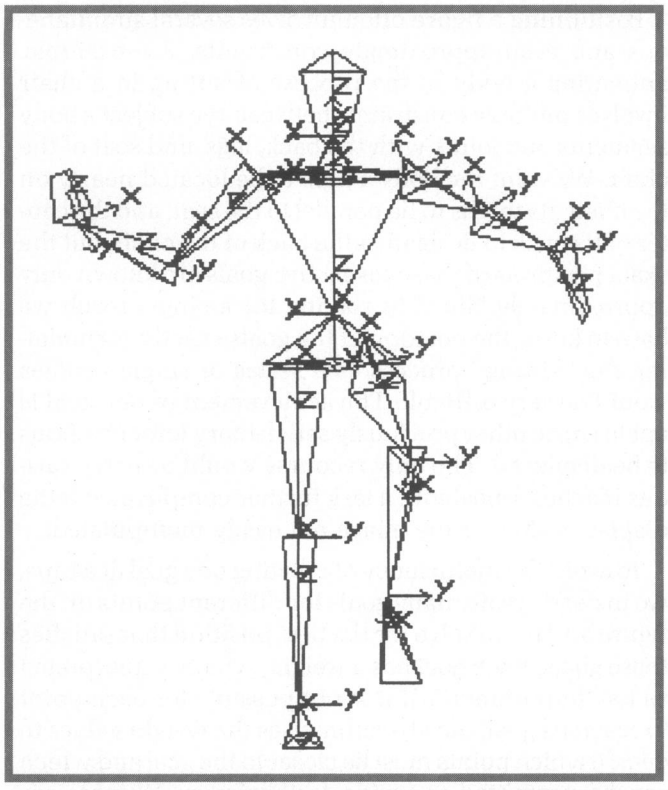


Figure 1. Each segment can be marked by its coordinate system during interactive manipulation.

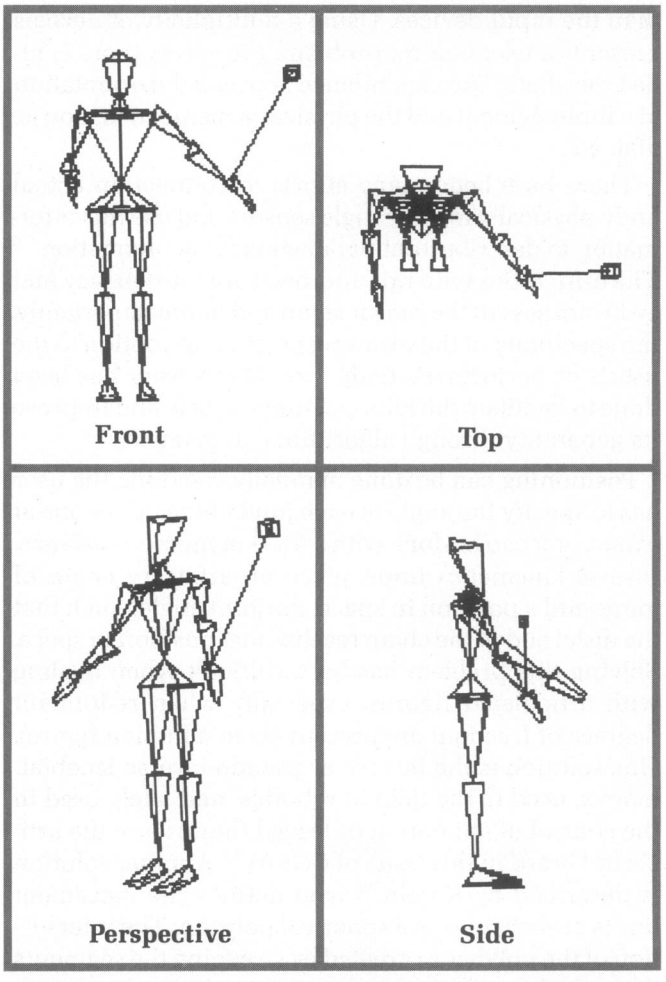


Figure 2. Four simultaneous views: three orthogonal and one perspective. The perspective view is defined by the user with the Polhemus as the camera.

articulated figure can be oriented using the Polhemus (see Figure 1). The three rotational degrees of freedom provided allow the user to establish segment orientation directly. The Polhemus is also used to set a goal for a joint using only the positional information (x, y, z). To make the task of positioning goals for joints easier for the user, POSIT provides the user with four different views of the body and the goal (see Figure 2). To get a better feeling for the position of the body and goals in space, the Polhemus can be used as a camera to look at the body and goals from different points of view.

Some examples will show how the multiple-constraint algorithm works. When the user sets only one goal for the figure, the algorithm acts as a simple reach algorithm and will move the body until the specified segment reaches its desired goal (Figures 3 and 4). When there are two simultaneous goals and they are both reachable, it will act as if two simple reaches have been executed on the body (Figure 5). When there are two or more goals and not all of them are reachable, the algorithm has to decide which segments are to be closer to their goals and which goals can be farther away. The decision is made using the goal weight. For example, if the decision involves two goals of values $G^1 = 40$ and $G^2 = 10$, the distance $d(S^1, G^1)$ (between segment 1 and goal 1) will be four times smaller than the distance $d(S^2, G^2)$ (see Figures 6 and 7).

Returning to the sitting problem, we see that it can be solved by setting four constraints or goals. The first and strongest goal must be from the lower end of the torso to the back of the seat; this goal is the strongest since it is the most important part of sitting down on a chair (for example, value = 100). Next we need a goal for the back of the figure to the back of the chair (for example, value = 10). To make the thighs parallel to the seat we need two goals from the knees to the front of the seat (for example, value = 10). Figures 8 and 9 show the possible position of goals and the solved position of the figure.

Two points should be noticed in this example. One is the difference in the values of the goals: The value of the goal at the lower torso is much higher than the others summed together to ensure that the position of the other goals will not affect the position of the lower torso. The goals for the upper torso and the knees are not positioned exactly, but they are set in the desired direction of that segment. Even though these goals are impossible to achieve, they are set such that they will affect the orientation and direction of their relative segments.

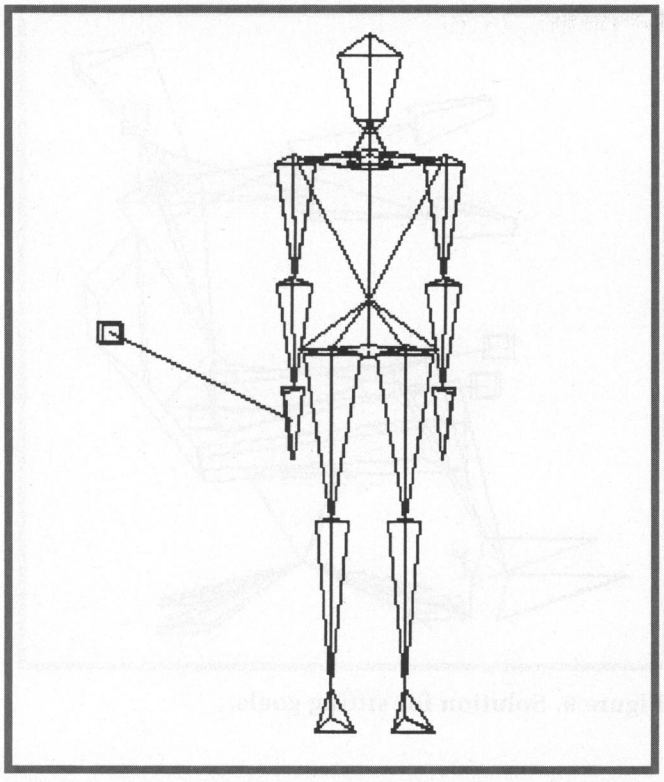


Figure 3. Goals for each segment are represented by a cube and are connected to their respective segments by a line.

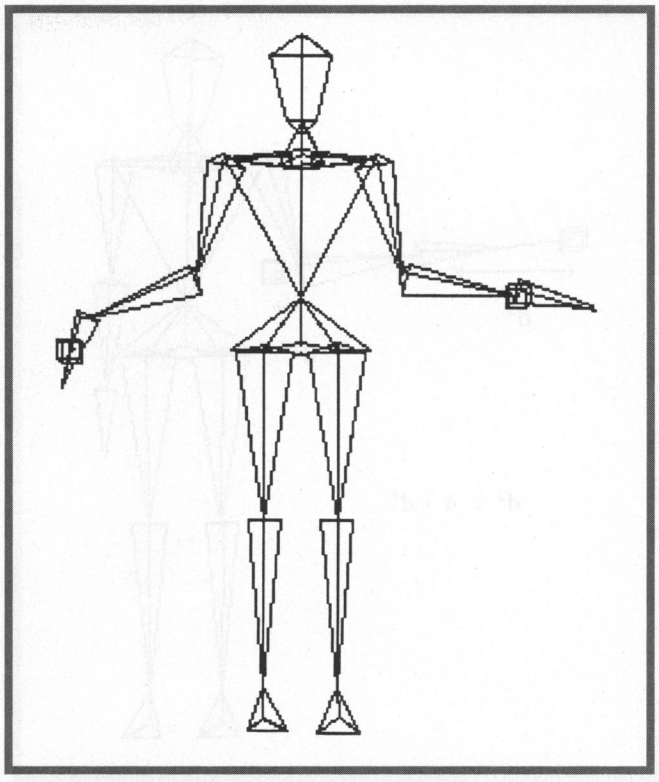


Figure 5. A solution to two reachable goals.

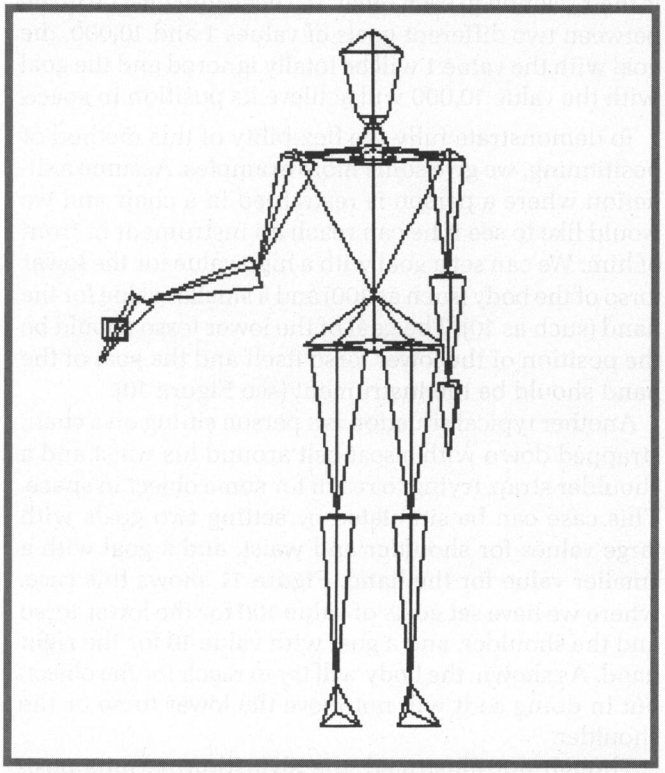


Figure 4. Solution to a single goal set for the right hand.

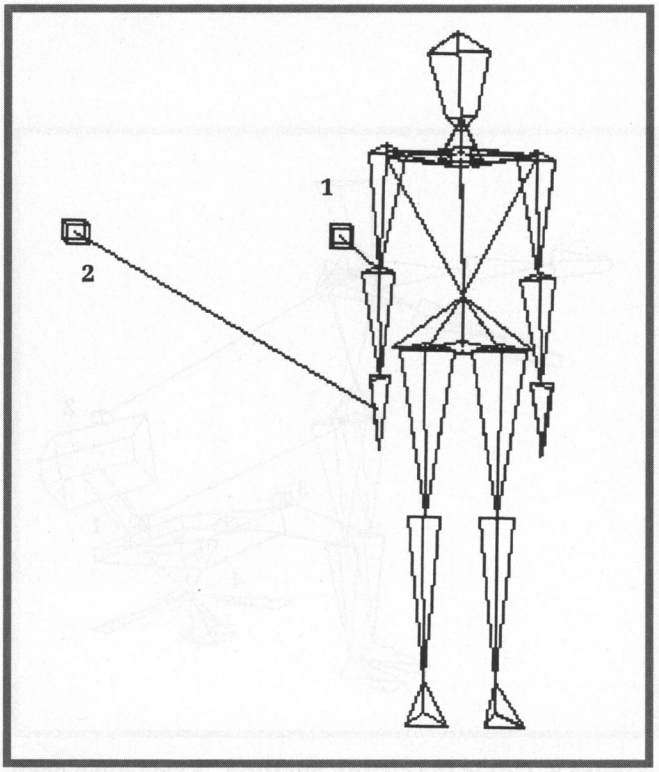


Figure 6. Two unreachable goals set for the right hand and upper arm: value 1 = 40, value 2 = 10.

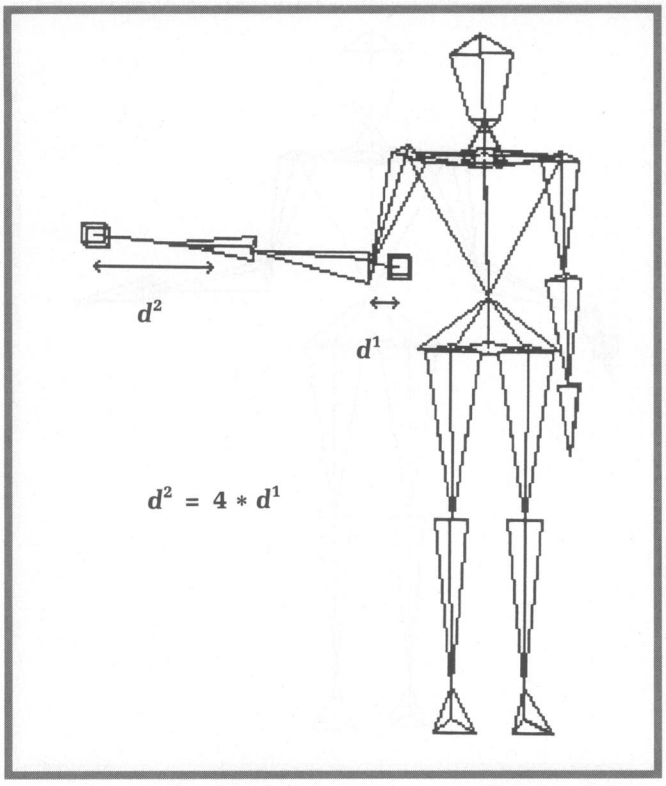


Figure 7. A possible solution. Notice the difference between d^1 and d^2: $d^2 = 4 * d^1$.

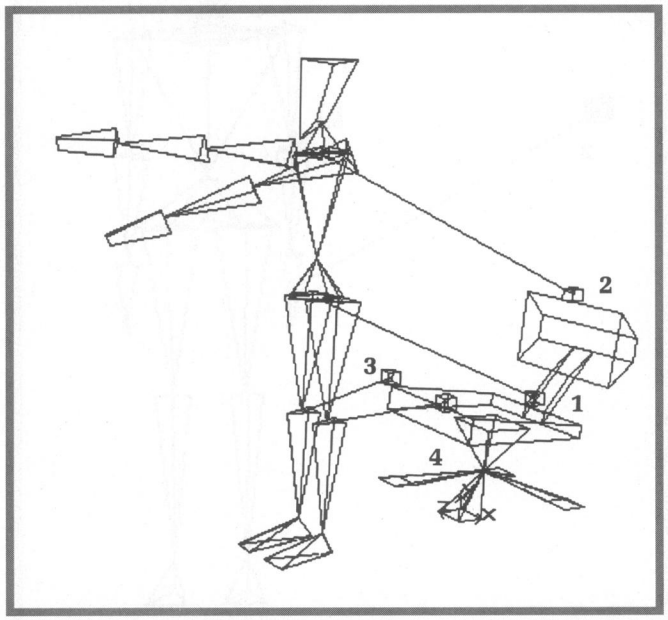


Figure 8. Goals for sitting position. Arms have been moved up for clarity: value 1 = 100, values 2, 3, 4 = 10.

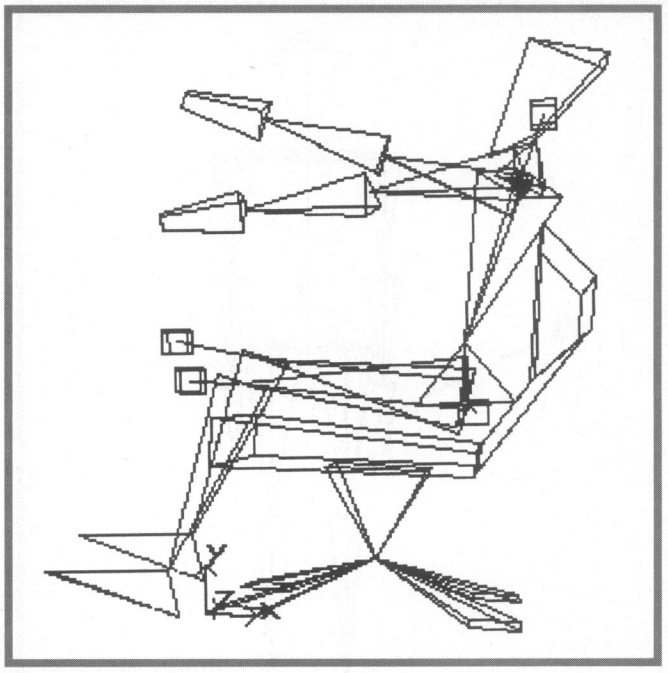


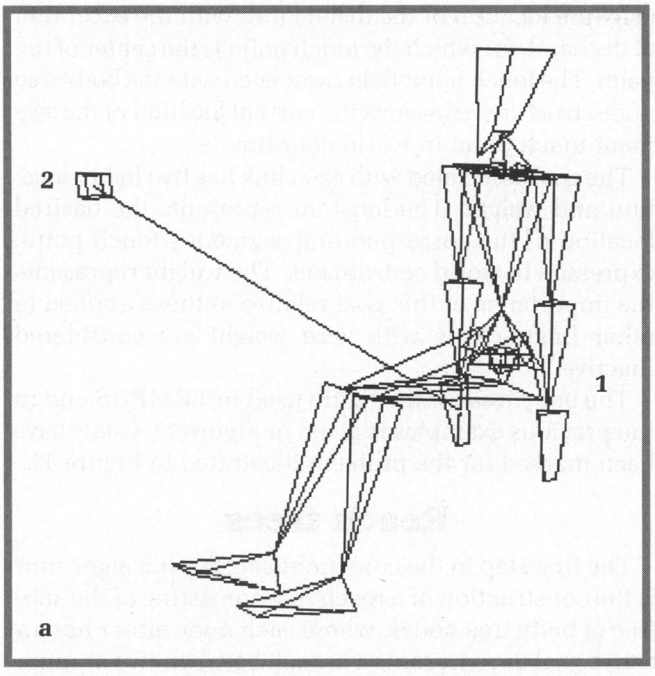
Figure 9. Solution for sitting goals.

There is no specified or preferred range for the value of goals. What is important in determining these values is their relation to each other. If a decision must be made between two different goals of values 1 and 10,000, the goal with the value 1 will be totally ignored and the goal with the value 10,000 will achieve its position in space.

To demonstrate fully the flexibility of this method of positioning, we give some more examples. Assume a situation where a person is restrained in a chair and we would like to see if he can reach an instrument in front of him. We can set a goal with a high value for the lower torso of the body (such as 100) and a smaller value for the hand (such as 10). The goal of the lower torso should be the position of the lower torso itself and the goal of the hand should be the instrument (see Figure 10).

Another typical situation is a person sitting on a chair, strapped down with a seat belt around his waist and a shoulder strap, trying to reach for some object in space. This case can be simulated by setting two goals with large values for shoulder and waist, and a goal with a smaller value for the hand. Figure 11 shows this case, where we have set goals of value 100 for the lower torso and the shoulder, and a goal with value 10 for the right hand. As shown, the body will try to reach for the object, but in doing so it will not move the lower torso or the shoulder.

Though not illustrated, the algorithm permits positions that are difficult to achieve by other means. For example, a two-handed reach goal can be specified by



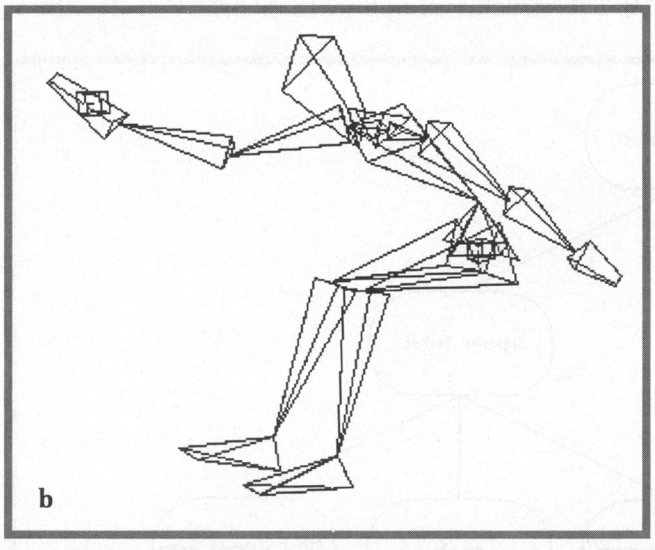


Figure 10. A large-value goal is set for the lower torso and a small-value goal for the hand: (a) value 1 = 100, value 2 = 10; (b) movement completed.

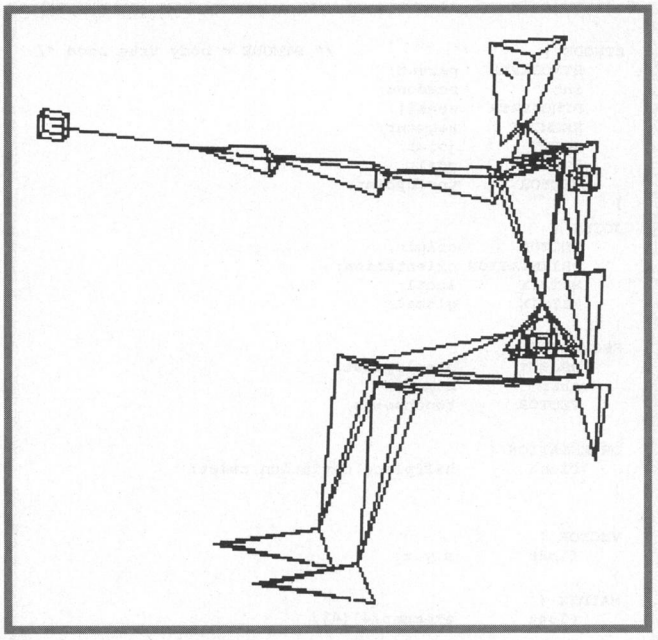


Figure 11. Two large-value goals set for the lower torso and the shoulder, and a small-value goal set for the hand.

giving each hand segment the same spatial goal. Once the goals are achieved, moving other joint goals will leave the two-handed reach intact up to the interpretation of the weight values. If the ankles are pulled in the opposite direction, the body will straighten out and maintain the goals as well as possible.

Body representation

The body is represented by a hierarchical tree where each node represents one body link (segment). A node contains the associated segment shape and the joint connecting the link to its parent. One link is chosen to be the body tree root; in POSIT the lower torso is used because it is typically nearest the body center of mass. Body trees can be constructed to model any type of articulated figure. The data structure used to represent the body tree is given in Figure 12.

Internal body links are connected by revolute joints. The joint structure associated with each node contains the origin of the joint (a fixed point expressed in its parent's coordinate frame) and the relative orientation of the two links. The local matrix associated with each joint represents the transformation from the parent link coordinate frame to the current link frame, which consists of a translation to the origin of the joint followed by a rotation in the displaced frame to achieve the relative orientation. The global matrix represents the position and orientation of each link in world coordinates. Prismatic joints can be modeled using this representation by fixing the relative orientation of the two frames and manipulating the origin.

The parent of the body root is considered to be the world coordinate system. The associated joint contains its position and orientation in the world frame, and both the local and global matrices represent the corresponding transformation.

The segment structure contains a filename, a precompiled display list that draws the segment in its own coordinate system, and a touch point. The touch point is the position in the segment's own coordinate system for which goals or constraints may be specified. This is typi-

```
BTNODE {                          /* BTNODE = body tree node */
    BTNODEPTR   parent;
    int         numsons;
    BTNODEPTR   sons[];
    SEGMENT     segment;
    JOINT       joint;
    GOAL        goal;
    VECTOR      touchpoint;
}
JOINT {
    VECTOR      origin;
    ORIENTATION orientation;
    MATRIX      local;
    MATRIX      global;
}
SEGMENT {
    OBJECT      displaylist;
    char        name[];
    VECTOR      touchpoint;
}
ORIENTATION {
    float       halfplane,deviation,twist;
}

VECTOR {
    float       x,y,z;
}
MATRIX {
    float       element[4][4];
```

Figure 12. Data structure used to represent the body tree.

cally the location of the distal joint, with the exception of the hand, for which the touch point is the center of the palm. The touch point field associated with the body tree node structure represents the current location of the segment touch point in world coordinates.

The goal associated with each link has two fields, location and weight. The location represents the desired location of the corresponding segment's touch point, expressed in world coordinates. The weight represents the importance of this goal relative to those applied to other links. Goals with zero weight are considered inactive.

The body tree for the figure used in TEMPUS and in the previous examples is given in Figure 13. Goals have been marked for the problem illustrated in Figure 11.

Reach trees

The first step in the constraint-satisfaction algorithm is the construction of a reach tree consisting of the subtree of body tree nodes, where each node either has an active goal or represents a branch between two or more subtrees that contain nodes with active goals (see Figure 14). This structure simplifies the satisfaction algorithm

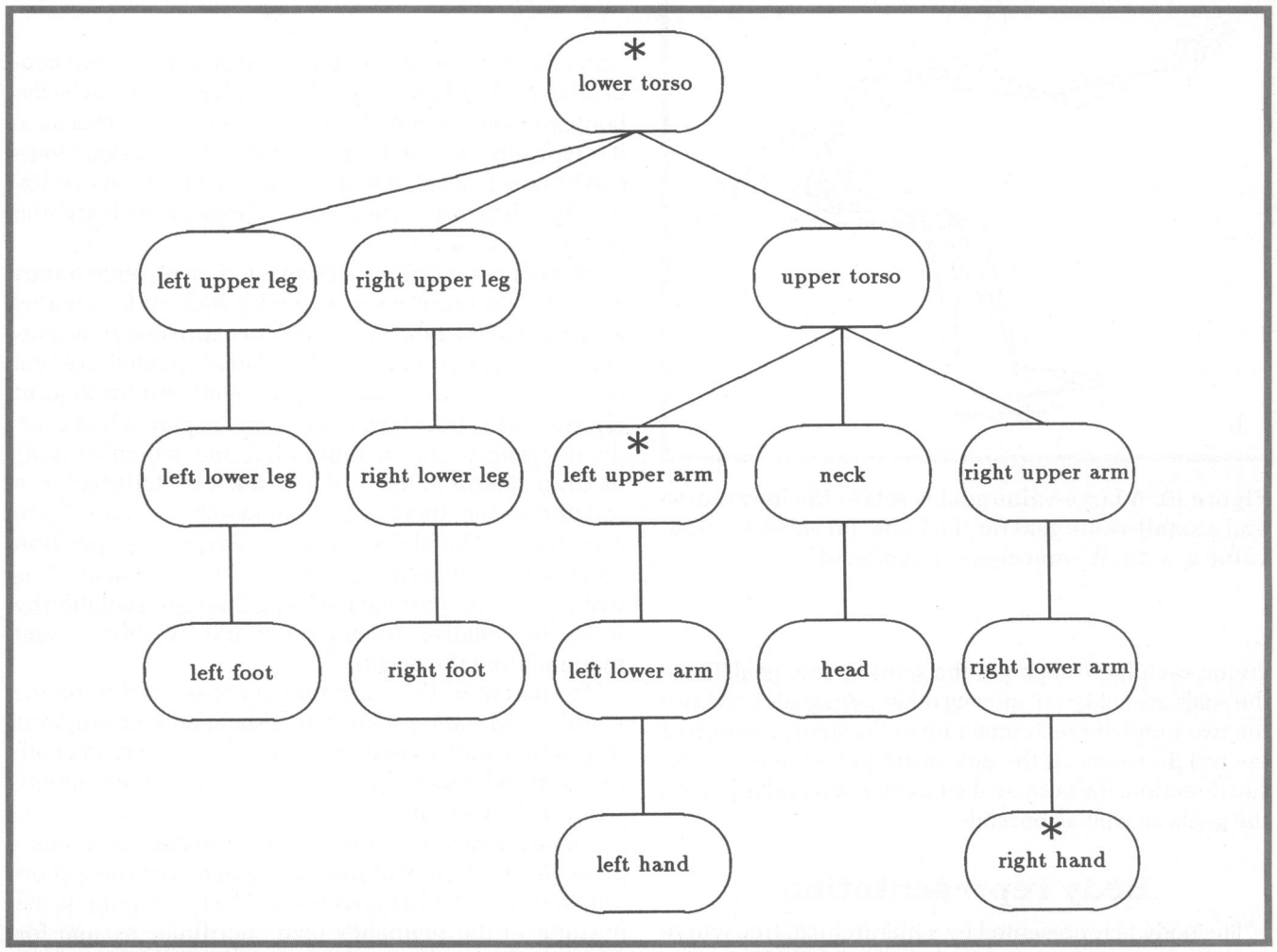


Figure 13. TEMPUS figure body tree: segments with goals are marked with *'s.

by identifying simple chains of links in the body tree. These chains may be manipulated without affecting other links with active goals that are not contained in the subtree rooted at the end of the chain. Each node in the reach tree is connected to its parent by a simple chain in the body tree. The tree is constructed by pruning all subtrees of the reach tree that do not contain nodes with active goals and then contracting all simple chains to have the end node the direct descendant of the start node.

In the sample body tree in Figure 15a, goals are active at nodes A, E, and G. The corresponding reach tree (Figure 15b) contains nodes A, B, E, and G. The linkages from A to B, B to E, and B to G are simple chains. Although node C represents a branch, only one of its subtrees contains a node with an active goal. The subtree connected to node F can be moved while solving the simple chain B, E without affecting any goals. In contrast, node B (which does not have an active goal) must be included in the reach tree because the chain A, E cannot be solved without moving the subtree B, D, G, which would affect the goal active at G. Although solving the simple chain A, B will affect the goals at E and G, these goals are completely contained in the subtree rooted at B and so will be solved recursively, as described below. Notice that this new structure has identified a constraint on the position of node B that is not directly related to a goal specified by the user, but is derived from the need to evaluate the interaction of the goals placed on nodes E and G.

This procedure is related to the "scope trees" described by Badler, O'Rourke, and Kaufman[18] to handle overlapping positioning instructions. The importance of this observation is that it frees positioning specification from the rigidity of the typical tree-structured hierarchy found throughout computer graphics. Rather than force positioning transformations to be nested (and hence described) relative to the ancestor node, the constraint positioning method evaluates joint positions (goals) in such a way that the joint transformations are derived from the spatial configurations required by the user. (The transformations are, of course, ultimately stored in the usual nested hierarchical fashion.) The "overlapping" scope of the constraining relations simply means that some joint constraints will affect other joint positions and vice versa.

To address the problem of a person reaching while restrained at the waist and shoulder, illustrated in Figure 11, goals are assigned to the lower torso, left shoulder, and right hand, as indicated in Figure 13. The resulting reach tree, shown in Figure 16, contains the three previously mentioned segments and the upper torso, which represents a branch between the goals on the shoulder and the hand. As with node B in the previous example, the upper torso is included in the tree implicitly for the purpose of balancing the two arm goals, as described in the next section.

```
RTNODE{
    RTNODEPTR   parent;
    RTNODEPTR   sons[];
    BTNODEPTR   btnode;    /* associated body tree node */
    VECTOR      lwd;       /* local weighted displacement */
    VECTOR      stwd       /* subtree weighted displacement */
    float       stweight;  /* total subtree weight */
```

Figure 14. Data structure for the reach tree.

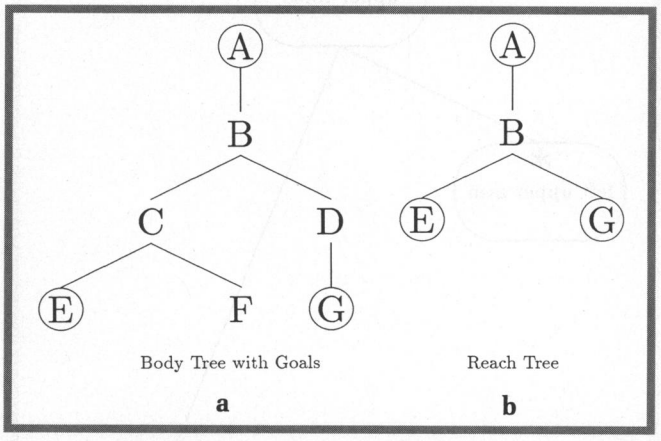


Figure 15. Sample body tree with goals (a) and reach tree (b).

Balanced reach trees

Associated with each node in the reach tree are two weighted displacements. The local weighted displacement or *lwd* is the displacement of the current link's touch point from its goal, scaled by the goal weight. The subtree-weighted displacement or *stwd* is the sum of the local weighted displacements for all of the nodes in the subtree rooted at the current node, including the displacement of the current node itself. This subtree displacement indicates the direction in which to move the current node to better satisfy its goal and the goals of its children, while observing the relative importance of the goals.

With this notion of reach tree weight, a *balanced reach tree* is a reach tree in which the root's subtree-weighted displacement has zero magnitude. This occurs when all the goals in the subtree are reached or when the distance of each node from its goal is balanced with the others according to given weights. In either of these situations, the tree is considered to be *perfectly* balanced. A reach tree can also be *optimally* balanced, when the subtree displacement has a nonzero magnitude, but the root node cannot be moved to decrease it without moving its parent. In such a case the tree may later be perfectly balanced after the parent is moved, as described next.

Reach tree balancing algorithm

The constraint-satisfaction algorithm used in POSIT is precisely the process of perfectly balancing the reach

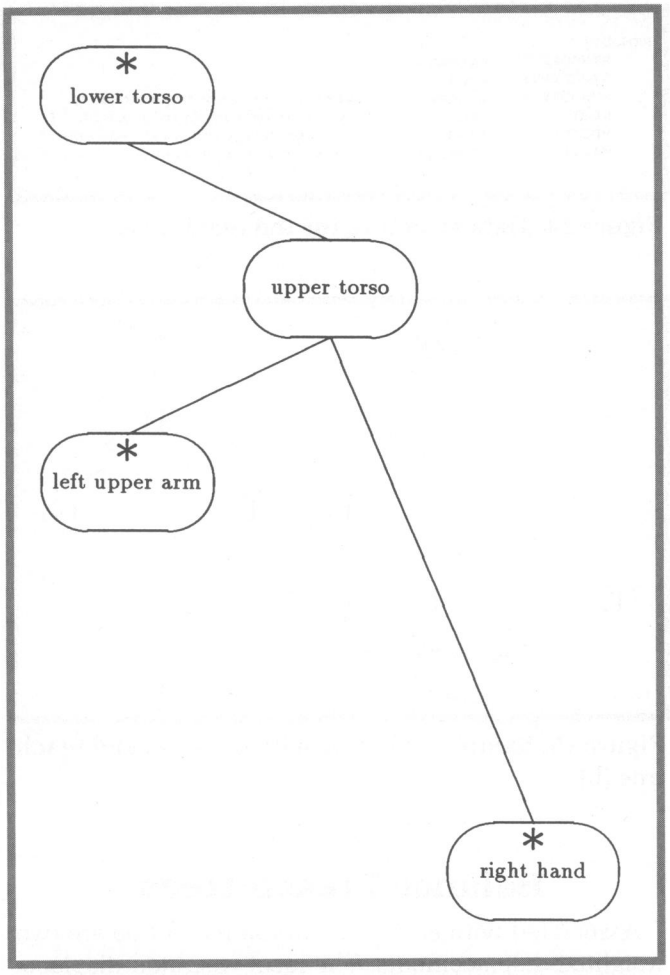


Figure 16. Reach tree for shoulder and waist restraint problem.

tree associated with the current set of constraints. This is accomplished via the following iterative algorithm, which calls itself recursively:

1. Recursively balance the reach trees rooted at each of the direct descendants of the current reach tree root.
2. Determine the subtree-weighted displacement for the current tree.
3. If displacement has zero magnitude or is no smaller than previous iteration then quit; else continue.
4. Determine a temporary goal for position of the root of the current tree.
5. Call an inverse kinematic solution to manipulate the simple chain from the root to its parent to place the root at this new position.
6. Repeat.

If the root of the current tree is also the root of the reach tree, its parent is considered to be the world coordinate system. There is no chain to solve; the segment is simply placed in the desired location, translating the entire tree by the current displacement. This will always result in a subtree displacement with zero magnitude, yet does not cause the algorithm to stop because the children must be recursively solved again before the subtree displacement is evaluated. The algorithm stops only when the displacement is zero or unchanged after the children have been repositioned.

Note that the root of the reach tree may not always be the body root, but rather a body tree node that is both qualified to be included in the reach tree and closest to the root. In such a case the linkage from the root of the reach tree to the body root cannot be changed by the constraint-satisfaction algorithm at all. If this is an undesirable artifact, then a constraint must be placed on the body root to ensure that it is considered in the solution.

Figure 17 gives the algorithm in pseudocode.

Solve_simple_chain

Solve_simple_chain currently uses a greedy algorithm based on the triangle inequality. Consecutive links are aligned until there is sufficient length to solve the reach as a triangle or the chain is fully straightened.

This approach works satisfactorily in the current implementation; however, some undesirable results may occur when joint angle limits are introduced. In particular, the greedy approach may stop at local minima when a better solution may exist. Establishing additional goals or changing goal weights solves the problem.

Extensions

We are investigating numerous extensions to POSIT. The elegance of the constraint-satisfaction algorithm itself makes the addition of other kinds of constraints possible. In particular,

- Joint angle limits and orientation constraints should be added.
- The user interface is being improved to take advantage of the experiences we gained in using the Polhemus as the principal input device. For example, the interface can use some subset of the possible Polhemus inputs at one time, rather than all of them, and provide better relative positioning.
- The simple reach might be solved by more powerful methods such as pseudo-inverse Jacobian.
- Constraint goals should be specifiable from environment objects or other parts of the figure.
- Points internal to a segment (that is, not just the distal joint) should be subject to constraints.
- Constraints should be allowed to have degrees of freedom so that, for example, a positional constraint can indicate contact with the floor plane but not care *where* on the floor the contact is actually made.[19]
- Constraints should be specified with respect to arbitrary coordinate systems, not just the global world space.[19]

```
balance_reach_tree(node:RTNODEPTR)
{

    /* balance the children of the current node and evaluate the best
       location at which to place the current root */

    clear_displacements(node);
    balance_children(node);
    approx_solution = get_approx_solution(node);

    repeat {

        /* remember current magnitude of stwd */

        prev_distance = mag(node->stwd);

        /* solve chain to parent in order to move current root */

        solve_simple_chain(node,node->parent,approx_solution);

        /* re-evaluate stwd */

        clear_displacements(node);
        balance_children(node);
        approx_solution = get_approx_solution(node);

    } n times or until (mag(node->stwd) >= prev_dist)
                                /* until displacement
                                   is not decreased or
                                   iterative limit is
                                   reached. */


}
clear_displacements(node)
{
    node->stwd     = 0, 0, 0;
    node->stweight = 0;
}
balance_children(node:RTNODEPTR)
{
    for (i = 0 to numsons) {
        balance_reach_tree(node->son[i]);
        node->stwd     += node->son.stwd;
        node->stweight += node->son.stweight;
    }
}
get_approx_solution(node)
{
    get_local_weighted_displacment(node);
    node->stwd     += node->lwd;
    node->stweight += node->btnode->goal.weight;
    return(node->btnode->touchpoint + stwd / stweight)
}
get_local_weighted_displacement(node);
{
    transform(node->btnode->segment.touchpoint,
              node->btnode->joint.matrix,
              node->btnode->touchpoint);
    node->lwd = (node->btnode->goal.location -
                 node->btnode->touchpoint) *
                 node->goal.weight;
```

Figure 17. The algorithm in pseudocode.

In addition, the springlike formulation of the constraint-satisfaction procedure leads to a simple interface for a true force-based dynamics simulation.[20] We are connecting a dynamics simulation into the POSIT interface to permit the interactive specification of constrained kinematics as well as individual joint torques and moments.

Conclusion

The combination of fast display, a six-axis input device, and multiple-constraint positioning assistance has made the task of positioning faster, easier, and more natural than before. The method used to solve for a simple reach is not a very efficient model, and it does not guarantee the best solution, especially with joint limits. Nevertheless, it does provide a reasonable position in a very modest time frame. All the examples shown here execute in 5 to 30 seconds on the Iris workstation. The efficiency is dependent upon the number of active constraints and the lengths of the moves required to achieve them. POSIT has demonstrated, however, that the task of positioning an articulated figure need not be as tedious as manually adjusting joint angles; rather, it can be as easy as visually establishing multiple goals and letting a straightforward tree-traversal algorithm achieve simultaneous satisfaction of all constraints. ■

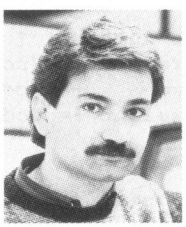
Acknowledgments

This research is partially supported by NASA Contract NAS9-17239, Lockheed Engineering and Management Services, NSF CER Grant MCS-82-19196, NSF Grants IST-86-12984 and DMC-85-16114, and ARO Grant DAAG29-84-K-0061, including participation by the US Army Human Engineering Laboratory.

References

1. T.J. O'Donnell and A.J. Olsen, "GRAMPS—A Graphics Language Interpreter for Real-Time, Interactive, Three-Dimensional Picture Editing and Animation," *Computer Graphics* (Proc. SIGGRAPH 81), Aug. 1981, pp. 133-142.
2. D.H.U. Kochanek and R.H. Bartels, "Interpolating Splines with Local Tension, Continuity, and Bias Control," *Computer Graphics* (Proc. SIGGRAPH 84), July 1984, pp. 33-41.
3. S. Steketee and N.I. Badler, "Parametric Keyframe Interpolation Incorporating Kinetic Adjustment and Phrasing Control," *Computer Graphics* (Proc. SIGGRAPH 85), July 1985, pp. 255-262.
4. N.I. Badler et al., "Positioning and Animating Human Figures in a Task-Oriented Environment," *The Visual Computer*, Dec. 1985, pp. 212-220.
5. T. Calvert, J. Chapman, and A. Patla, "The Integration of Subjective and Objective Data in the Animation of Human Movement," *Computer Graphics* (Proc. SIGGRAPH 80), July 1980, pp. 198-203.
6. C.M. Ginsberg and D. Maxwell, "Graphical Marionette," in *Motion: Representation and Perception*, N.I. Badler and J. Tsotsos, eds., North-Holland, New York, 1986, pp. 303-310.
7. R. Paul, *Robot Manipulators: Mathematics, Programming, and Control*, MIT Press, Cambridge, Mass., 1981, pp. 85-117.
8. M. Girard and A.A. Maciejewski, "Computational Modeling for the Computer Animation of Legged Figures," *Computer Graphics* (Proc. SIGGRAPH 85), July 1985, pp. 263-270.
9. M. Girard, "Interactive Design of 3D Computer-Animated Legged Animal Motion," *CG&A*, June 1987, pp. 39-51.
10. J.U. Korein, *A Geometric Investigation of Reach*, MIT Press, Cambridge, Mass., 1985.
11. I.E. Sutherland, "SKETCHPAD: A Man-Machine Graphical Communication System," SJCC, Spartan Books, Baltimore, Md., 1963, p. 329.
12. J. O'Rourke and N.I. Badler, "Model-Based Image Analysis of Human Motion Using Constraint Propagation," *IEEE Trans. Pattern Analysis and Machine Intelligence*, Nov. 1980, pp. 522-536.
13. A. Marion, K. Fleischer, and M. Vickers, "Toward Expressive Animation for Interactive Characters," *Proc. Graphics Interface 84*, Canadian Information Processing Soc., Toronto, 1984, pp. 17-20.
14. J.F. Coblentz, P. Gueneau, and N. Bonjour, "Computerized Determination of Optimal Posture," *Biostereometrics Proc.*, SPIE, Bellingham, Wash., 1985, pp. 244-247.
15. K.H. Manoochehri, *Articulated Figure Positioning by Multiple Constraints and 6-Axis Input*, master's thesis, Univ. of Pennsylvania, Philadelphia, 1986.
16. C. Schmandt, "Spatial Input/Display Correspondence in a Stereoscopic Computer Graphic Work Station," *Computer Graphics*, (Proc. SIGGRAPH 83), July 1983, pp. 253-261.
17. D. Baraff and N.I. Badler, "Handwaving in Computer Graphics: Efficient Methods for Interactive Input Using a Six-Axis Digitizer," tech. report, Dept. of Computer and Information Science, Univ. of Pennsylvania, Philadelphia, 1986.
18. N.I. Badler, J. O'Rourke, and B. Kaufman, "Special Problems in Human Movement Simulation," *Computer Graphics* (Proc. SIGGRAPH 80), July 1980, pp. 189-197.
19. N.I. Badler et al., "Animation Using Constraint-Based Kinematics," tech. report, Dept. of Computer and Information Science, Univ. of Pennsylvania, Philadelphia, 1987.
20. B. Paul and R. Schaffa, "DYSPAM User's Manual," Dept. of Mechanical Eng. and Applied Mechanics, Univ. of Pennsylvania, Philadelphia, 1986.

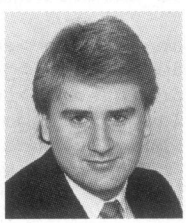
Norman I. Badler's biographical information appears on page 11 in the "Guest Editor's Introduction."



Kamran Manoochehri is a computer graphics software specialist for Cranston/Csuri Productions, Inc., where he is responsible for designing and developing software. His interests include character animation and new techniques for digitizing 3D objects.

Manoochehri received a bachelor's degree with honors in computer and information science from Temple University. He received an MS in computer and information science from the University of Pennsylvania. While attending the University of Pennsylvania on a research fellowship, he developed an interactive paint program for the generation of textures and systems for graphical display of articulated bodies. For his thesis, he developed a real-time system that positions articulated figures by multiple constraints.

Manoochehri is member of Phi Beta Kappa and ACM.



Graham Walters is a research fellow at the University of Pennsylvania, where he received a BS in computer science and engineering in 1985. He is now working on an MS in computer and information science with a computer graphics major, which he expects to receive in May 1987. His interests lie in animation, modeling, and image synthesis.

Walters is a member of ACM SIGGRAPH.

Badler and Walters can be reached at the Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104-6389. Manoochehri's address is Cranston/Csuri Productions, Inc., 1501 Neil Ave., Columbus, OH 43201.