Other vehicles used in factories and warehouses simply follow a reflective tape track on the floor or sense the presence of a trail laid down with invisible inks and chemicals.

That "computers were the technological revolution of this modern age" was an understatement for their evolution into mobile servants will more than confirm such a statement.

We had better prepare; for . . .

The Robots are coming!

BIBLIOGRAPHY

General References

R. Malone, *The Robot Book,* Push Pin Press, New York, 1978

A. Silverstein and V. Silverstein, *The Robots are Here,* Prentice-Hall, N.J., 1983

P. Berger, *Robot Catalogue,* Dodd, Mead & Company, New York, 1984. Chapter 2 has an excellent survey of entertainment robots.

H. Geduld and R. Gottesman, Robots-Robots-Robots, Little, Brown and Company, New York, 1978.

J. Reichardt, Robots, Viking Press, New York, 1978.

P. Marsh, *Robots,* Salamander Books, UK, 1985.

B. Krasnoff, Robots: Reel to Real, Arco Publishing, New York, 1982

Design Quarterly, *Robots,* MIT Press, Cambridge, Mass., 1983

R. M. Hefley, *Starlog Robot Guide,* Starlog Press, New York, 1979. Provides excellent reference for robots in films and television.

SIMULATORS, GRAPHIC

C. A. KLEIN
A. A. MACIEJEWSKI
Ohio State University
Columbus, Ohio

PURPOSE OF USING GRAPHIC ROBOTIC SIMULATION

There are many situations in which a computer simulation with a graphic display can be very useful in the design of a robotic system. First of all, when a robot is planned for an industrial application, there are many commercially available arms that can be selected. A graphics-based simulation would allow the manufacturing engineer to evaluate alternative choices quickly and easily (1). The engineer can also use such a simulation tool to design interactively the workcell in which the robot operates and integrate the robot with other systems, such as part feeders and conveyors with which it must closely work. Even before the the workcell is assembled or the arm first arrives, the engineer can optimize the placement of the robot with respect to the fixtures it must reach and ensure that the arm is not blocked by supports. By being able to evaluate workcell designs off-line and away from the factory floor, changes can be made without hindering factory production and thus the net productivity of the design effort can be increased.

Once a robot is installed, graphic simulations are very valuable in the generation and verification of trajectory plans. By viewing a proposed trajectory on a computer screen, many errors can be discovered that would have been dangerous for the real robot to attempt to perform. A programming or teaching error could cause the actual arm to strike the end effector into a work table, another part of the arm, fixturing, or a human teaching the arm with a pendant. Collision avoidance, though, involves more than simply preventing the end effector from striking an object in the work space. Often, without a graphics simulation, it is hard to anticipate which parts of the arm, such as protruding portions, may hit supports. Even when the robot is present, it may be preferable to program one leg of a trajectory path, view the simulation on a screen, and then actually perform it (2).

Beyond safety considerations, graphic simulations are also used to optimize motion trajectories. Trajectories can be designed so that no joint is pushed to a kinematic limit or exceeds its maximum velocity or acceleration. Plans can be interactively modified to minimize travel time and increase manufacturing throughput. A complete simulation will verify that when an arm is reaching for parts stacked in a pallet, for example, it can reach the most distant part at the required orientation. Automatically generated plans can be previewed.

Where multiple arms are at work sharing a common workspace, the need for a careful graphics simulation is even greater (3). Such a design makes greater use of common facilities such as feeders and fixtures, but requires more careful planning. The two arms must be coordinated, and each must be considered as a dynamically moving obstacle with respect to the other arm (4).

The mechanical designer can advantageously use a graphics simulation in designing arms. Computer graphics can show the workspace of a given geometric configuration without even requiring a built model. The location of singularities can be determined and modified. The control engineer can use graphic simulations to choose motors and a control system design for a new robot. Graphics output is a desirable part of dynamic simulations, since a visual display can provide more information to the designer than would printed numerical values. Without simulation tools, many systems would have been impossible to design. For example, the Space Shuttle arm is incapable of operating in a full gravitational field and was instead designed, to a large extent, based on simulations (5).

Lastly, the value of graphic simulation cannot be underestimated in education and research. Many of the principles of robotic control can be taught through simulation, utilizing realistic graphics to relatively large groups of students inexpensively and safely. New robotic techniques are being developed by researchers through simulation long before the actual hardware systems are tested.

COMPONENTS OF A ROBOTIC SIMULATION SYSTEM

Depending on the application, there are many possible configurations for a robotic graphic simulator system in terms of the type of display, mode of entering data, method of saving the output, the computer hardware doing the processing, and the software controlling the system. Choices depend on the degree of realism required, whether the simulation must run in real time, the amount and type of interaction desired, and the amount of resources that can be devoted to the system.

Displays can be based on either vector or raster display

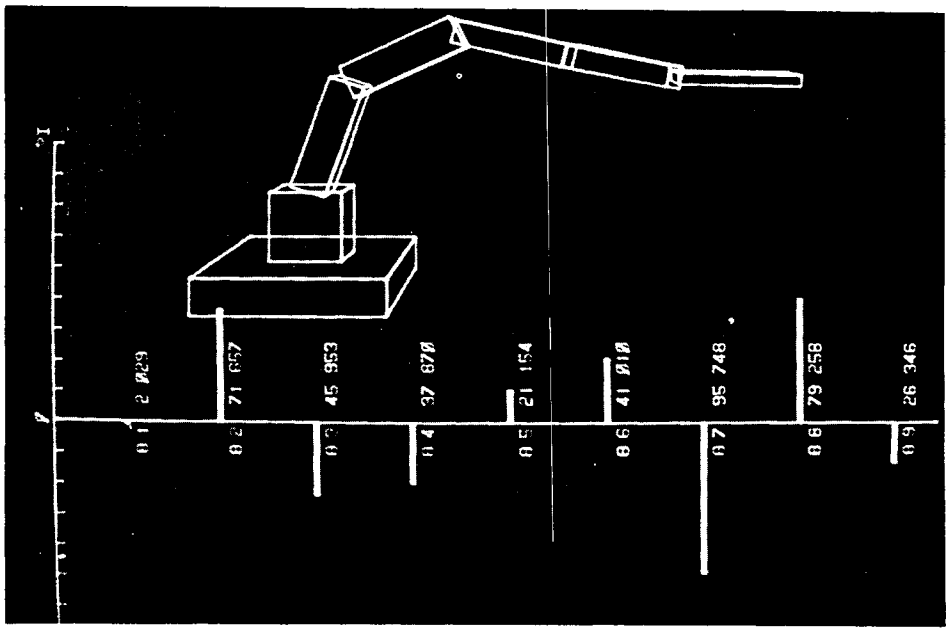

Figure 1. Real-time vector display of a new robot geometry under development. Superimposed on the robot image is a dynamic display of joint angles to allow the designer to evaluate automatically generated trajectories.

devices. Vector displays draw lines at arbitrary angles on a screen and can quickly draw outlines of robotic systems. These devices can be further subdivided into vector refresh displays and storage-tube systems. In a vector refresh display, the points are held in a memory and are constantly being drawn on the screen. Figure 1 illustrates an application where the features of this type of display are desirable. Vector refresh displays allow smooth animation, but have the disadvantage of flickering when a large number of points are on the screen. Storage tubes accumulate an image in the screen itself and eliminate the flicker even for very complicated drawings. Except for a "write-through" mode where a weaker image can be drawn, as on a vector refresh terminal, animation is only possible by erasing the entire screen and redrawing. Raster displays, on the other hand, are like conventional television screens. Raster displays can be used to show solid-color-shaded representations of objects, but require more computer processing to generate a complete image. Figure 2 shows an application (6) where such a solid representation is important in evaluating clearance between objects. Raster displays, of course, can also be used to draw wire-frame skeletons of robot images. Some simulation systems may use both vector and raster displays, a vector display to view a simulation in real or near real time and a raster display to generate realistic images for later documentation.

For recording the graphic output of a graphic simulation, a number of devices are available. Plotters are ideal for saving the images of vector displays. Some of the new laser printers can be programmed to act as a plotter, and some can draw shaded polygons and thus permit a realistic display. Of course, for raster color-shaded graphics, film, video tapes, or laser disks can be used for recording animations. An additional output of some simulators is a file of robot motion commands. Although the operator may specify the desired motion in a format independent of the robot manufacturer, some systems, in effect, compile this specification to one that the particular robot can use later.

A number of input devices are used to enter commands and graphical data into simulators. Virtually all systems include keyboards to enter information. A device such as a digitizer, mouse, trackball, or a light pen can also be used to enter graphical information. Often the graphic input device is used to enter commands by moving a cursor on the computer screen to a labeled area. The user then activates that choice by some action such as pressing a button on a mouse or hitting a car-

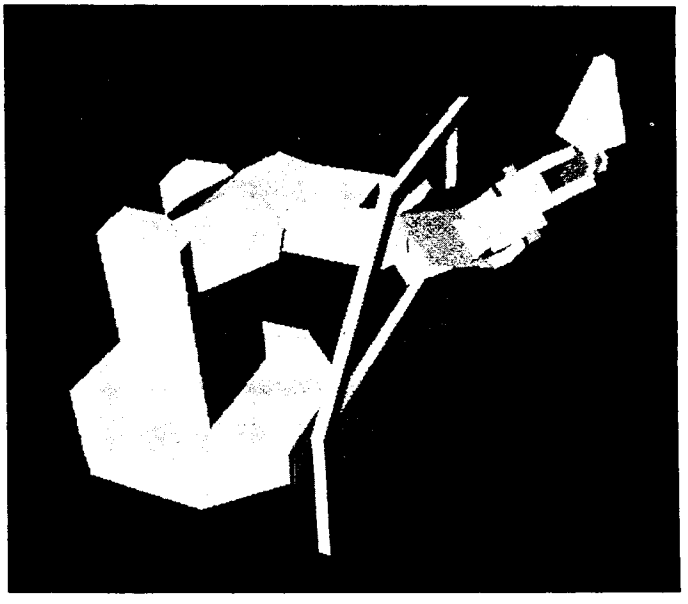

Figure 2. A more realistic color-shaded image of a manipulator displayed on a raster device. This robot has the same kinematic geometry as the one illustrated in Figure 1, but shows a more complete physical description of the links. One of the purposes of this simulation was to evaluate an automatic obstacle-avoidance scheme for the guidance of an arm through the window of a car door (6). Courtesy of *The International Journal of Robotics Research,* MIT Press.

riage return key. Some systems include data bases of alternative robot geometrics, thus greatly reducing the amount of graphic data that needs to be manually entered.

A wide range of computer hardware supports graphic simulations. Simulators run on micro-, mini-, and mainframe computers. Figure 3 provides an example of a commercially available microprocessor-based system. The display may be a graphics terminal connected to a host through a serial line. Display processors can be more sophisticated and may be connected to a host by parallel ports with DMA (direct memory access) access to the host's memory. Display processors include high performance vector displays with real-time update capabilities. Some raster systems contain frame buffers, which are memory arrays of dimension of the screen resolution and contain the color data for each pixel on the screen. Many systems add local processing power to do polygen fill, vector generation, character and circle generation, zoom, pan, and so on. Several systems are based on special-purpose VLSI (very large scale integration) scale chips that have been designed to perform quickly computations needed in graphics (7). Some systems combine processing power and memory to do hardware depth sorting, ie, polygons in three-dimensional space are properly displayed so that closer polygons obscure more distant ones, independently of the order in which they are drawn. Some graphics controllers reside in the backplane of a personal computer (PC) and share memory with the host microprocessor. Other graphic simulations run on engineering workstations. Such systems are usually based on powerful computer processors, often have sophisticated graphics, and are designed for a single user. Whereas many of these systems run a variety of software products, some are designed to be a complete turnkey system, as a dedicated tool for a particular analysis or design problem, such as mechanical CAD/CAM.

Similarly, software varies considerably for different robotic simulators. Many robot designers and researchers have written their own software systems to meet the needs of their requirements and have explained the basic principles through publications (8–10). In some cases, software is available specifically for robot simulation; in others, it is within the capabilities of other software systems such as mechanical CAD/CAM packages. For many systems, the graphics can be used as a final output, verifying programming, operation, or control specified by some robot-control or simulation language (11). Some new systems are designed in the framework of expert systems and allow the user to specify the robotic structure in an object-oriented programming style (12).

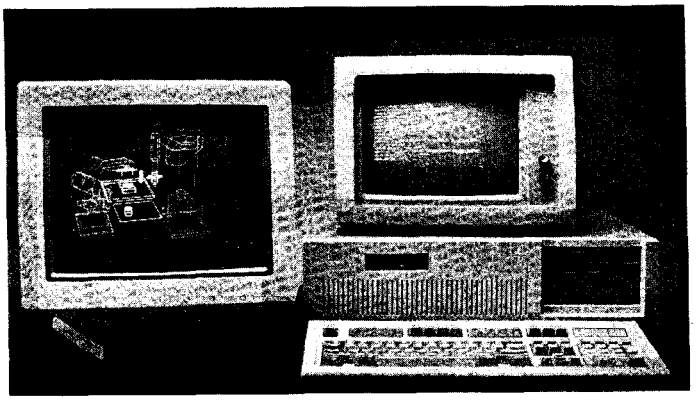

Figure 3. The CimStation interactive workcell modeling system by SILMA, showing their computer-graphic-simulation option running on a personal computer. Courtesy of SILMA, Inc., Los Altos, Calif.

ROBOT MODELING

In this section, the theory behind robot simulations is discussed. There are two general types of robot simulations: kinematic and dynamic (see also KINEMATICS). Kinematic simulations show the motion of a manipulator without regard to how those motions are achieved by actuator forces and torques, whereas dynamic simulations consider masses, forces, and torques to calculate the resulting motion (13). If it is known that the robot has been designed to meet certain kinematic specifications, then the production engineer does not need to consider the systems dynamics explicitly and can instead concentrate on motion planning. Even when a dynamic simulation is done, the resulting motion can be specified kinematically and then displayed with a kinematic-based graphics system.

The modeling of a robot system can be divided into a functional description of its motion and a physical description of its actual geometry. The first part of the model can be divided into a kinematic description of the robot itself and a means of trajectory planning to achieve a desired motion. The second part is concerned with modeling and three-dimensional characteristics of the robot's shape.

Kinematics

In order to simulate the motion of a robot, the kinematic structure of the particular robot must first be defined. This structure is usually described by a notation like that developed by Denavit and Hartenberg (14). Using this notation, each degree of freedom within an articulated robot is assigned a unique coordinate system, with the relationship between adjacent coordinate systems defined by four parameters. The degrees of freedom of the robot, either rotary or prismatic, are referred to as joints and the interconnecting portions are called links. The four parameters used to specify the relationships between coordinate systems are the length of the link a, the twist of the link α, the distance between links d, and the angle between links θ. The definition of these parameters for both rotary and prismatic degrees of freedom is illustrated in Figure 4. A simple procedure for defining the origins of the various coordinate systems is given in Ref. 15. More complicated joints, such as ball-and-socket joints, can be modeled mathematically as combinations of simple joints. It should be noted that these parameters only specify the kinematic structure of a robot and do not describe the physical appearance of its links.

Given the above specification of coordinate frames, the relationship between adjacent coordinate frames is given by a rotation of θ, followed by translations of d and a, and a final rotation of α. By combining these transformations, it can be shown (15) that the relationship between adjacent coordinate frames i and $i - 1$, denoted by ${}^{i-1}A_i$ is given by the homogeneous transformation matrix

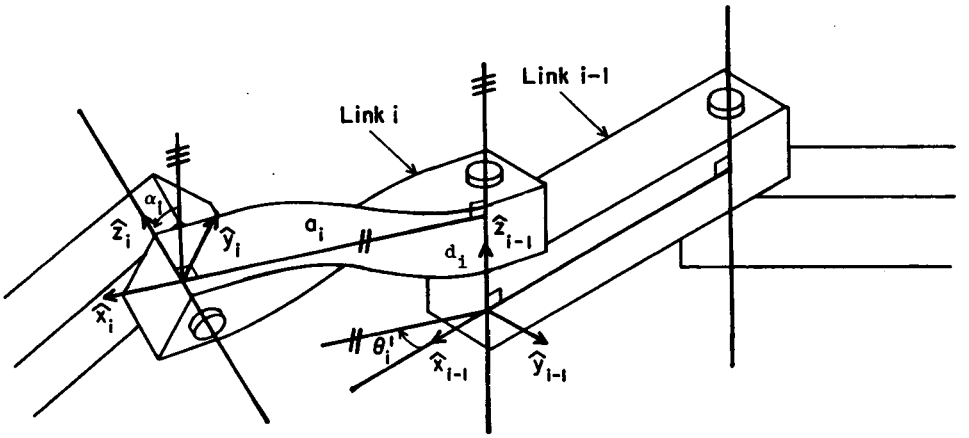

Figure 4. The definition of the Denavit and Hartenberg parameters for describing the kinematic relationship between robot links. The joint rotation angle, as depicted, is negative. Courtesy of Pergamon Press.

$$ {}^{i-1}A_i = \begin{bmatrix} R_{3\times3} & p_{3\times1} \\ 0 \;\; 0 \;\; 0 & 1 \end{bmatrix}. \tag{1} $$

where

$$ R = \begin{bmatrix} \cos\theta_i & -\cos\alpha_i \sin\theta_i & \sin\alpha_i \sin\theta_i \\ \sin\theta_i & \cos\alpha_i \cos\theta_i & -\sin\alpha_i \cos\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i \end{bmatrix} \tag{2} $$

and

$$ p = \begin{bmatrix} a_i \cos\theta_i \\ a_i \sin\theta_i \\ d_i \end{bmatrix}. \tag{3} $$

The above homogeneous transformation matrix has an easily visualized physical interpretation. The submatrix R, which is sometimes called the rotation matrix, is composed of the direction cosines between the two related coordinate systems; thus it represents the three-dimensional rotation required to align the two coordinate systems. The vector **p** is a position vector, which specifies the difference in position between the origins of the two coordinate systems.

By multiplying adjacent link transformations, the homogeneous transformation between any two coordinate systems i and j may be computed as follows:

$$ {}^iA_j = {}^iA_{i+1}\,{}^{i+1}A_{i+2} \cdots {}^{j-1}A_j \tag{4} $$

Using the above equation, the position and orientation of a robot's end effector can be computed given the values of the joint variables, ie, θ for rotary joints and d for prismatic joints. This computation is typically referred to as the direct kinematics problem. Note, however, that when applying robots for the completion of some useful task, the reverse problem needs to be solved. That is, given the position and orientation of the robot's end effector, the required joint values need to be found. This problem, referred to as the inverse kinematics problem, is not as easily solved (15).

Trajectory Planning

The solution of the inverse kinematics problem usually relies on the evaluation of inverse trigonometric functions and is restricted to particular robot geometries. A general closed-form solution for an arbitrary robot structure has not been found. However, for those geometries for which solutions do exist, inverse kinematics represents the simplest method for controlling robots used in pick-and-place tasks. For tasks described in these terms, such as some assembly and materials handling tasks, only selected configurations of the end effector are important for successful task completion. Paths between such configurations are unconstrained, except for such global considerations such as collision avoidance. Motion planning for such cases can consist of inverse kinematics for selected configurations with joint interpolation in between. In other cases, some Cartesian control between configurations may be required. Thus straight-line motion, combined with one- or two-axis rotations between configurations, is popularly employed (16,17).

In a growing number of applications, however, the above inverse kinematics techniques are not sufficient. When used for arc welding or paint spraying, for example, tool paths required for successful completion are not only based on a fixed set of positions and orientations, but must be controlled along particular trajectories at specified rates. In order to simulate motion planning at this level, the concept of resolved-motion rate control was developed (18). Essential to this concept is the Jacobian matrix. The Jacobian matrix J relates the velocity of the robot's end effector to the joint variable velocities through the equation

$$ J\dot{\theta} = \begin{bmatrix} \dot{x} \\ \omega \end{bmatrix} \tag{5} $$

where $\dot{x}$ is a three-dimensional vector defining the translational velocity, ω is a three-dimensional vector defining the rotational velocity, and $\dot{\theta}$ is an n-dimensional vector representing the joint velocities, n being the number of degrees of freedom that the robot possesses. Although a number of techniques

for calculating the Jacobian have been studied (19), a particularly elegant and efficient method for graphic simulators is available that only uses the position vector p and one column of the rotation matrix R of the homogeneous transformations ${}^0A_i$, for $i = 1$–n (20). In this application of screw-axis variables, the only computation required, other than that required for the homogeneous transformations, is a single cross-product per column of J. This formulation, therefore, is particularly useful for graphic simulators, since the individual homogeneous transformation matrices must already be computed for generating a display of the robot.

Thus for tasks specified as desired end-effector velocities, the required joint velocities to achieve the task are obtained by solving the linear set of equations given by equation 5. The desired joint velocities are given by

$$\dot{\theta} = J^{-1} \begin{bmatrix} \dot{x} \\ \omega \end{bmatrix} \tag{6}$$

if J is square and nonsingular. For those cases where the number of degrees of freedom do not match the dimension of the specified velocity or if J is singular, J^{-1} is not defined. In these cases, even though the inverse of the Jacobian does not exist, there do exit generalized inverses or other techniques that provide useful solutions to equation 6 (21–23).

Geometric Representation

The kinematic specification of robots described above is sufficient for defining motion; however, it says nothing about the physical structure of a robot, ie, what are the shapes of the various links, actuators, and so on. These qualities, which are important for checking for such critical features as collisions as well as providing a realistic graphic display, require geometric modeling techniques (24). There exist a variety of ways to represent the shape of three-dimensional objects, but they all belong to basically one of two groups: boundary or surface representations, and solid representations.

Surface representations, as the name implies, consist of two-dimensional primitives which are used to define the boundary or surface of the solid object to be modeled. There is no explicit modeling of the solid composing the object other than that implied by a closed surface. Whereas there are a number of different primitives used for describing surfaces, planar polygons are the most frequently employed. This is due to the fact that planes can be described by linear equations which greatly simplify many of the algorithms required to display objects (see the following sections). Thus a solid three-dimensional object can be modeled by a collection of polygons which are used to approximate its surface. These polygons are usually described by the positions of their vertexes. A typical data format for objects described in the above manner is seen in Figure 5. The variables i and j are integers that correspond to the number of points and the number of polygons, respectively, used to describe the object. The next i rows of numbers corresponds to the x, y, and z coordinates of the points. The last j rows of numbers are integers that define the vertexes of the polygons. The first column contains the number of vertexes for that particular polygon, with the remaining columns specifying the vertexes by an index into the point list. Since each vertex coordinate is defined only once and is referenced in a polygon by its index, a considerable amount of storage is saved over explicitly listing the coordinates in each polygon. Additional parameters can be added to describe such properties as color, reflectance, etc, which are then used by the illumination model.

i – number of points
j – number of polygons

$$\left.\begin{matrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \vdots & \vdots & \vdots \\ x_i & y_i & z_i \end{matrix}\right\} \text{points}$$

$$\left.\begin{matrix} k_1 & v_{1,1} & v_{1,2} & \cdots & v_{1,k_1} \\ k_2 & v_{2,1} & v_{2,2} & \cdots & v_{2,k_2} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ k_j & v_{j,1} & v_{j,2} & \cdots & v_{j,k_j} \end{matrix}\right\} \text{polygons}$$

Figure 5. A typical data-file format for the description of objects defined by polygonal surfaces.

The difficulty with describing object surfaces as a collection of planar polygons is that they provide a poor approximation to curved surfaces. For this reason, primitives called parametric surfaces or patches, which are usually described by higher order polynomial equations, are often used (25). Of these, bicubic equations are commonly employed since they allow first-order continuity between adjacent patches, resulting in smooth composite surfaces. It should be noted, however, that the same features of parametric surfaces that bring flexibility to surface modeling also result in complexity in display algorithms. Although display algorithms that use parametric surfaces directly do exist, it is common to first subdivide patches into a collection of polygons and then deal with the simpler linear surfaces.

In contrast to surface representations, solid representations explicitly describe the interior as well as the boundary of objects. One such representation is constructive solid geometry, where an object is defined by Boolean operations on a set of solid primitives (26). The solid primitives used are usually restricted to very simple shapes and therefore have difficulty in modeling free-form solids. In contrast, probably the most flexible solid representation is the three-dimensional generalization of the parametric surface known as a hyperpatch (27). Once again, however, this flexibility results in computational complexity for the display algorithms so that this representation is only used when such flexibility is essential.

COMPUTER GRAPHICS

Viewing Transformations

One of the major purposes of computer graphics is to simulate the three-dimensional characteristics of an object visually, by displaying what that object would look like from various different points of view. In order to describe the relationship between the viewer and the objects to be simulated, it is convenient to define three parameters that specify this relationship: the eyepoint (EP), which is the simulated position of the viewer; the center of interest (COI), which is the point at which

the viewer's eyes are directed; and the view angle (VA), which defines the cone of vision. Having defined these three parameters, the positions of objects described in a world data-base coordinate system can now be related to the coordinates of a particular graphics output device through the use of homogeneous transformations.

The first of these transformations, which is typically referred to as a point-of-view transformation, aligns the directions of the x, y, and z axes to those of the output device. This resulting coordinate system is sometimes referred to as eyespace coordinates. The homogeneous transformation relating coordinates in worldspace to eyespace is given by

$$\begin{bmatrix} \cos\theta & 0 & -\sin\theta & -\mathrm{COI}_x\cos\theta + \mathrm{COI}_z\sin\theta \\ -\sin\theta\sin\phi & \cos\phi & -\cos\theta\sin\phi & \mathrm{COI}_x\sin\theta\sin\phi - \mathrm{COI}_y\cos\phi + \mathrm{COI}_z\cos\theta\sin\phi \\ -\sin\theta\cos\phi & -\sin\phi & -\cos\theta\cos\phi & \mathrm{COI}_x\sin\theta\cos\phi + \mathrm{COI}_y\sin\phi + \mathrm{COI}_z\cos\theta\cos\phi + |v| \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

with

$$\cos\theta = \frac{v_z}{\sqrt{v_x^2 + v_z^2}}$$

$$\sin\theta = \frac{v_x}{\sqrt{v_x^2 + v_z^2}}$$

$$\cos\phi = \frac{\sqrt{v_x^2 + v_z^2}}{|v|}$$

$$\cos\phi = \frac{v_y}{|v|}$$

where v is the vector from the EP to the COI.

The next transformation that is typically applied is a perspective transform, which results in realistic depth perception. It is convenient to represent this transform in the following form:

$$\begin{bmatrix} \cot\frac{\mathrm{VA}}{2} & 0 & 0 & 0 \\ 0 & \cot\frac{\mathrm{VA}}{2} & 0 & 0 \\ 0 & 0 & \frac{z_{\max}}{z_{\max} - z_{\min}} & \frac{z_{\min} z_{\max}}{z_{\min} - z_{\max}} \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

This transform is in a device-independent format, with screen coordinates being in the range -1 to $+1$ in the horizontal x and the vertical y directions, and 0–1 in the depth z direction. This is particularly useful for simulators that possess multiple output devices of different resolutions. The values z_{min} and z_{max} are the minimum and maximum eyespace coordinate values, which are to be mapped to screen coordinates. The use of screen z coordinates in the context of hidden surface removal will be discussed in a following section. For display on a particular device, objects must only undergo one final transformation, which contains the device-dependent characteristics. This transformation includes such information as horizontal and vertical screen resolution, aspect ratio, depth resolution, and device-coordinate system position and orientation. Whereas it is possible to combine all three transformations by multiplying together the three corresponding matrices, the final screen transform is usually performed separately. The intermediate form of the object data after the perspective transform is then used for further processing, the first step of which involves clipping.

Clipping

Clipping is the process by which only the desired portions of objects are retained for display, with the others being "clipped." This is usually required since most output display devices have a limited range of coordinates; thus display coordinates must be limited to available values in order to prevent wraparound. The particular choices of the view angle, and z_{min} and z_{max} in the transformations discussed above, imply a volume of allowable coordinates in three-dimensional space. Due to the perspective transformation, this volume is in the shape of a truncated pyramid. An important step in all clipping algorithms is determining whether a point lies inside or outside of this volume. This problem is usually broken down into six subproblems, ie, whether a point lies on the inside of the six planes that define the truncated pyramid. It is important to note that even though these points have been multiplied by the perspective transformation matrix, normalization, ie, division by the homogeneous coordinate, should not be performed before clipping (28).

The choice of clipping algorithm depends to some extent on the form of the desired output, in particular, whether the output will be displayed as a line drawing or as shaded polygons. Even though there are a number of efficient algorithms for clipping objects described as a collection of line segments (29), there are advantages to using a more computationally expensive polygon clipping algorithm. In particular, since a polygonal format is most widely used for graphics data bases, a polygon clipper will preserve this format. In addition, clipped data in a polygonal format can be displayed equally well as a line drawing or a shaded image, thus retaining flexibility in output devices. The most popular polygon clipping algorithm is one developed by Sutherland and Hodgman (30). This algorithm, referred to as reentrant polygon clipping, defines a polygon to be a collection of vertexes that are checked against the six clipping planes in turn. Those vertexes that are inside are retained, whereas those on the outside are discarded, with additional vertexes added when a polygon edge intersects a clipping plane.

Hidden Surface Removal

Whereas the above clipping process guarantees that all remaining objects are within the defined viewing area, they are still not all visible on the display. A further processing step is needed to determine which objects or portions of objects are in front of and occluding others. This process, known as

hidden surface removal, has a number of different solutions, a taxonomy of which is presented in Ref. 31. The common characteristic of all algorithms is that they must sort objects in order to determine which ones are in front of others. One major difference between algorithms is based on whether this sorting is done in object space, that is, the coordinate space in which the objects are defined, or in image space, ie, the screen-coordinate system. The object-space algorithms are in general more precise; however, image-space algorithms enjoy an advantage in efficiency. Although no one algorithm is clearly superior over all others, the depth or z-buffer algorithm is currently the most widely implemented for graphic simulators.

The z-buffer algorithm is basically an extension of the frame-buffer concept. It is a simple hardware solution to hidden surface removal, which maintains an additional buffer that is used to store the current depth value of the closest object at each pixel. Thus depth sorting is trivially accomplished by checking an object's depth against that which is currently in the z-buffer. The chief disadvantage previously cited against this algorithm was the large amount of memory required to maintain this additional buffer. When considering the current trends in size and cost of high speed memory, however, these concerns are no longer a prohibitive factor. Additional disadvantages include aliasing effects, but extensions to the z-buffer algorithm have shown that these can be corrected as well (32). All in all, the z-buffer's simplicity and speed account for its popularity, especially in cases where real-time simulation is required.

Illumination Models

When dealing with solid-shaded graphic displays, an illumination model must be included in the simulation in order to determine the displayed intensity of objects. In the real world, the way an object appears is a result of various physical properties of the object, which determine how light is absorbed, transmitted, and reflected. Other factors that affect this appearance include properties of the light source and positional relationships among the object, viewer, and light sources. The extent to which these factors are included in the simulation determines the amount of realism achievable. The tradeoff, naturally, is that the more accurately these effects are represented in the computer model, the more computationally expensive it becomes. For this reason, there exists an entire range of illumination models that are used in graphic simulations.

Basic illumination models start with a simple model of diffuse illumination based on Lambert's law (29). This gives the reflected intensity of an object as being only a function of the incident-light direction and the object's surface normal. The modeling of ambient light is typically included as a constant term, except in the more refined models. The modeling of specular reflection can be included as an additive component by also considering the relative position of the viewer. Further refinements to the specular component can include modeling of the spatial distribution of the reflected light, as well as its wavelength dependence. The application of these models to objects defined by planar elements will result in shading discontinuities due to the discontinuity in surface normals. Whereas this may be of no concern for objects that actually do consist of planar surfaces, it presents difficulties when modeling smooth curved surfaces. Solutions to this problem, without increasing the number of polygons used to describe the surface, have been presented (33). By using bilinear interpolation on the intensities obtained at polygon vertexes, a smooth shaded surface will result. Improvements in this smooth-shading scheme can be made by interpolating the surface normals instead of the intensity at the expense of extra computation (34).

The common characteristic of the models discussed above is that they all use only local information, such as surface normals, incident-light direction, and line-of-sight direction, to determine an object's intensity. Although these models are the most common for graphic simulators due to their computational efficiency, models that incorporate global information can produce much more realistic images. The most common of these algorithms are based on ray tracing (35). Ray tracing is based on following a beam of light from the viewer's eye, through a pixel, and into the world in which the objects are defined. Calculations, based on which objects this beam intersects and how it is split due to reflection and refraction, are used to determine the intensity displayed at that pixel. This technique easily incorporates shadows, transparency, reflection, and refraction, and has resulted in the most realistic computer-generated images to date. Its major disadvantage, however, is that it is very computationally expensive and as such is typically employed for small numbers of high quality promotional photos.

Aliasing

When using raster display devices, the effects of aliasing should be considered if high quality images are desired. Aliasing effects, which are most often represented by the jagged or staircase appearance of straight lines, are an inherent part of fixed-period sampling. Algorithms that use such spatial sampling to determine the intensity of pixels on the screen will exhibit aliasing effects. Aliasing effects are particularly objectionable when viewed in animations, since the viewer readily notices this display artifact changing over time. Temporal aliasing can also be present in animations as a result of sampling in time. The fundamental principle in either type of sampling, spatial or temporal, is that frequencies greater than one-half of the sampling frequency cannot be distinguished and will appear to be composed of lower frequency "aliases." There are basically three approaches to solving the aliasing problem, or in other words, antialiasing.

The first approach to antialiasing is to increase the sampling frequency. Thus for a display device with a given resolution, instead of computing one intensity for each pixel, the image is computed at a higher resolution with the samples then averaged to provide the intensity for a pixel (36). Whereas this technique works reasonably well, it becomes impractical when a scene contains very high frequency components. A second approach is to limit the frequency components of objects within a scene to below half of the sampling frequency. This operation is known as low pass filtering or convolution (29). A simple method that is equivalent to low pass filtering is to consider pixels as consisting of finite areas instead of point samples. Filtering operations, however, are in general rather computationally expensive. A third approach to antialiasing is to avoid using fixed-period sampling. If stochastic techniques

are applied to the sampling process, it can be shown that aliases will not be present (37,38). The tradeoff is that instead of aliases, stochastic sampling introduces broad-band noise into the image. It appears, however, that this noise is psychologically less objectionable than the ordered error of aliases and is therefore a viable technique for antialiasing.

GRAPHIC SIMULATOR ORGANIZATION

This section is a discussion of how the above concepts are integrated into a complete system, with an overview of the software modules involved and an emphasis on their interaction. A diagram of a generic graphic-simulator system is presented in Figure 6, where various input and output databases and their associated program modules are defined. This figure is meant to be a functional diagram only and does not necessarily imply a physical separation between its elements. For example, the blocks of this figure commonly are coroutines in a large integrated system.

The first requirement of a robotic simulator is to have a means of specifying the robot to be simulated. Conceptually, a robot representation can be divided into two components: a functional definition of its kinematic parameters and associated data, such as joint limitations; and a physical description of the actual geometry of the robot's links, actuators, and so on. A software module that can define and modify the kinematic parameters of a robot is useful for determining such features as workspace limitations and is instrumental in the design process. The modeling of a robot's physical geometry is performed by standard mechanical CAD/CAM-type software. This modeling software serves a dual purpose since it is also used to model the desired final product, as well as fixtures with which the robot is to operate. A complete robot specification, functional and geometric, is often stored in a database library of commercially available robot systems. This simplifies the selection process and allows flexibility in evaluating the performance of prospective robots for a particular task or workcell arrangement. The capability of modifying these robot specifications, however, is still important since standard models are sometimes customized for special applications.

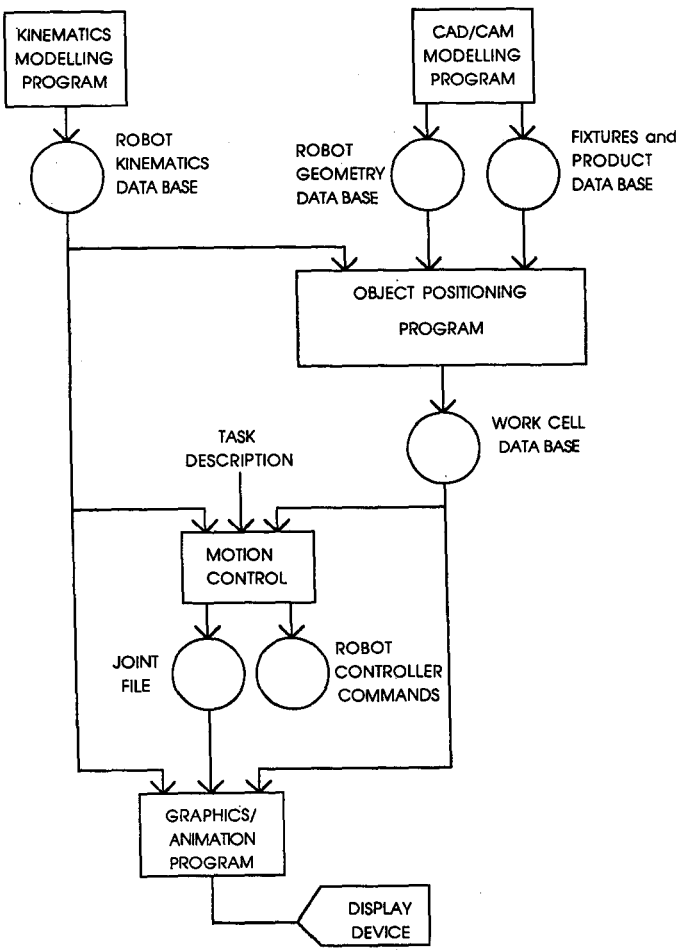

Figure 6. Organization of a general graphic-robotic-simulation system. Rectangles indicate software modules and circles indicate data.

Since one of the major applications of graphic simulators is to model the integration of robots into a manufacturing process, it is essential to have software available that can aid in the specification of workcell arrangements. The inputs into such a program are the complete robot description, fixtures and/or other machines that are required by the process, and a description of the product itself. The positioning of the robot and other movable objects and transport systems can then be specified. This process is to some extent an iterative procedure in which objects are placed, and then such constraints as the working envelope are checked until an optimum solution is achieved. This may require substitution of alternative robot geometries, illustrating the interaction with the software module described above.

The static positioning of objects within a workcell, however, is only a small part of its design. The major responsibility of robotic simulators is to model time-varying characteristics to ensure proper operation. This requires a complete geometric description of all objects within the workcell, including how they change over time; the kinematic, dynamic, and controller information for the robot being simulated; and a description of the task or process to be completed. This is the level that corresponds to teaching a physical robot. Robot simulation as a means of off-line programming has a number of advantages in terms of cost, time, and safety. The level at which the designer interacts with this module varies with the sophistication of the system. Some systems are menu-driven, which has a certain resemblance to teaching a robot with a pendant since discrete choices are made. However, a menu-driven system is more powerful since some options correspond to operations not easily performed with a physical manipulator. Alternatively, many simulators use a device-independent programming language, which is used to specify the robot's motion. This language can be of a relatively low level, which requires position information for the robot's end effector. More sophisticated languages, however, can take into account all of the geometric information available from the workcell and product descriptions, as well as process-control specifications, and independently produce end-effector trajectory information from high level commands. Once again, the optimization of motion control is more or less an iterative procedure, which may require modification of the robot's trajectory or the work-

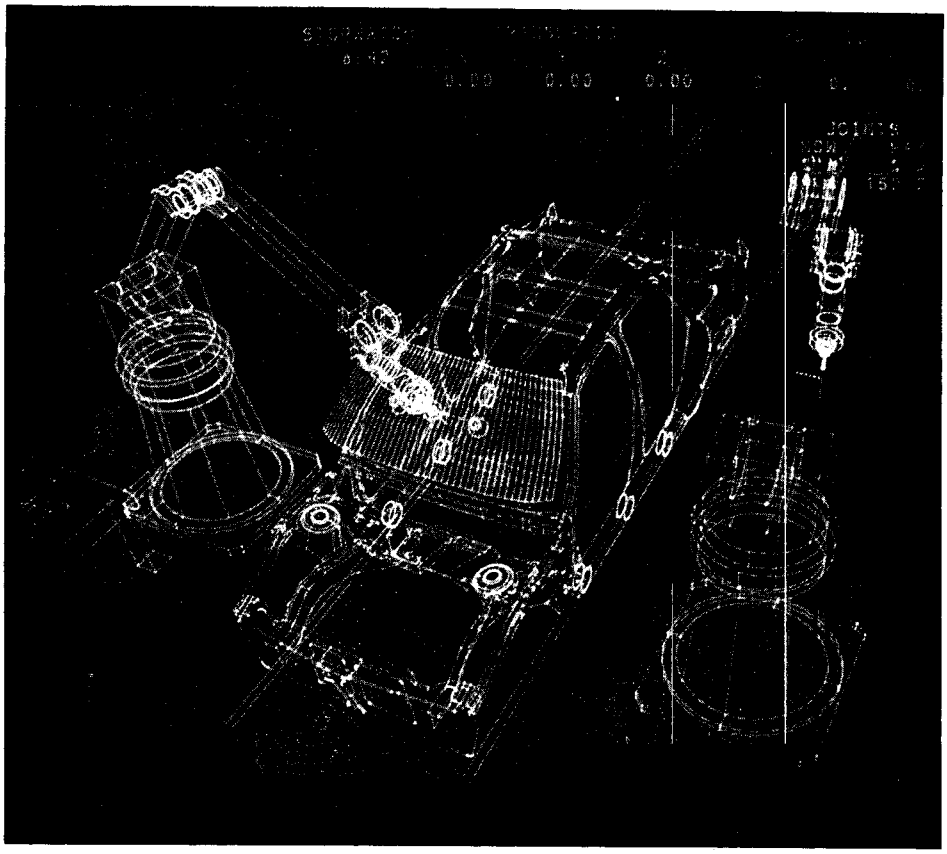

Figure 7. A graphic simulation display of a complete workcell with two arms cooperating on a product assembly line. This display is from McDonnell/Douglas using their integrated family of robot simulation systems. Courtesy of McDonnell/Douglas Manufacturing and Engineering Systems Company.

cell arrangement. It may be even necessary to go back and try a different robot geometry. Once the process is completed, most simulators can translate the motion-specification commands into a device-dependent controller format for downloading into the physical robot on the factory floor.

The final logical software component of the simulator system is used concurrently with all of the above procedures, namely, the actual graphic display and animation software. While performing the previously described design stages, the engineer uses computer graphics to view the progress of the design interactively. Saved data can also be postprocessed to achieve even more realistic representation for documentation. Graphic simulations give the designer the ability to take geometric models and display them in three dimensions, allowing the designer to look at them from any angle or any degree of detail. Thus it is used during the initial CAD/CAM modeling phase to provide direct visual feedback on product or robot specifications. In the next stage, it also provides the ideal interface for evaluating the workcell design. Figure 7 shows an example simulation of an entire workcell in operation. With the information on the time-changing variables from the motion-control module, it can produce real-time animations for detecting collisions between elements, optimizing end-effector motion, and generally checking the quality of the overall manufacturing process.

BIBLIOGRAPHY

1. M. Donner, "Computer Simulation to Aid Robot Selection" in A. Pugh, ed., *Robotic Technology,* Peter Peregrinus Ltd, London, UK, 1983, pp. 103–111.
2. R. Mahajan and J. S. Mogul, "An Interactive Graphic Robotics Instructional Program-I GRIP: A Study of Robot Motion and Workspace Constraints," *Robots* **8,** 41–56 (June 1984).
3. J. F. Callan, "The Simulation and Programming of Multiple-arm Robot Systems," *Robotics Eng.* **8** (4), 26–29 (Apr. 1986).
4. E. Freund and H. Hoyer, "Collision Avoidance in Multi-robot Systems" in H. Hanafusa and H. Inoue, eds., *Robotics Research: The Second International Symposium,* MIT Press, Cambridge, Mass., 1985, pp. 135–146.
5. A. K. Bejczy, "Sensors, Controls, and Man-machine Interface for Advanced Teleoperation," *Science,* **208,** 1327–1335 (June 20, 1980).
6. A. A. Maciejewski and C. A. Klein, "Obstacle Avoidance for Kinematically Redundant Manipulators in Dynamically Varying Environments," *Int. J. Robotic Res.* **4**(3), 109–117 (Fall 1985).
7. J. H. Clark, "The Geometry Engine: A VLSI Geometry System for Graphics," *Comput. Graphics* **16**(3), 127–133 (July 1982).
8. A. A. Maciejewski and C. A. Klein, "SAM—Animation Software for Simulating Articulated Motion," *Comput. Graphics* **9**(4), 383–391 (1985).

9. M. Hornick and B. Ravani, "Computer-aided Off-line Planning and Programming of Robot Motion," *Int. J. Robotics Res.* **4**(4), 18–31 (Winter 1986).
10. P. A. Fitzhorn and W. O. Troxell, "A Dynamic Approach to the Robotic Design Cycle" in *1986 Proc. IEEE Int. Conf. on Robotics and Automation,* San Francisco, Calif., Apr. 1986, pp. 353–358.
11. W. A. Gruver, B. I. Soroka, J. J. Craig, and T. L. Turner, "Industrial Robot Programming Languages: A Comparative Evaluation," *IEEE Trans. Syst. Man Cybernetics* **SMC-14**(4), 1–7 (July–Aug. 1984).
12. R. Dillmann and M. Huck, "A Software System for the Simulation of Robot Based Manufacturing Processes," *Robotics* **2**(1), 3–18 (Mar. 1986).
13. K. S. Fu, R. C. Gonzalez, and C. S. G. Lee, *Robotics: Control, Sensing, Vision, and Intelligence,* McGraw-Hill, Inc., New York, 1987.
14. J. Denavit and R. S. Hartenberg, "A Kinematic Notation for Lower Pair Mechanisms Based on Matrices," *ASME J. Appl. Mech.* **22**(2), 215–221 (June 1955).
15. R. Paul, *Robot Manipulators: Mathematics, Programming, and Control,* MIT Press, Cambridge, Mass., 1981.
16. R. Paul, "Manipulator Cartesian Path Control," *IEEE Trans. Syst. Man Cybernetics,* **SMC-9**(11), 702–711 (Nov. 1979).
17. R. H. Taylor, "Planning and Execution of Straight Line Manipulator Trajectories," *IBM J. Res. Dev.* **23**(4), 253–264 (July 1979).
18. D. Whitney, "The Mathematics of Coordinated Control of Prostheses and Manipulators," *J. Dyn. Syst. Meas. Control* **94**(4), 303–309 (Dec. 1972).
19. D. E. Orin and W. W. Schrader, "Efficient Jacobian Determination for Robot Manipulators" in M. Brady and R. Paul, eds., *Robotics Research: The First International Symposium,* MIT Press, Cambridge, Mass., 1984, pp. 727–734.
20. K. J. Waldron, "Geometrically Based Manipulator Rate Control Algorithms," *Mech. Mach. Theory,* **17**(6), 379–385 (1982).
21. C. A. Klein and C. H. Huang, "Review of Pseudoinverse Control for Use with Kinematically Redundant Manipulators," *IEEE Trans. Syst. Man Cybernetics* **SMC-13**(2), 245–250 (Mar.–Apr. 1983).
22. A. Ben-Israel and T. N. E. Greville, *Generalized Inverses: Theory and Applications,* Wiley-Interscience, New York, 1974.
23. T. L. Boullion and P. L. Odell, *Generalized Inverse Matrices,* Wiley-Interscience, New York, 1971.
24. M. E. Mortenson, *Geometric Modeling,* John Wiley & Sons, Inc., New York, 1985.
25. D. F. Rogers and J. A. Adams, *Mathematical Elements for Computer Graphics,* McGraw-Hill, Inc., New York, 1976.
26. A. A. G. Requicha, "Representations for Rigid Solids: Theory, Methods, and Systems," *Comput. Surv.* **12**(4), 437–464 (Dec. 1980).
27. M. S. Casale and E. L. Stanton, "An Overview of Analytic Solid Modeling," *IEEE Comput. Graphics Appl.* **5**(2), 45–56 (Feb. 1985).
28. J. F. Blinn and M. E. Newell, "Clipping Using Homogeneous Coordinates," *Comput. Graphics,* **12**(3), 245–251 (1978).
29. D. F. Rogers, *Procedural Elements for Computer Graphics,* McGraw-Hill, Inc.. New York, 1985
30. I. E. Sutherland and G. W. Hodgman, "Reentrant Polygon Clipping," *CACM* **17**(1), 32–42 (1974).
31. I. E. Sutherland, R. F. Sproul, and R. A. Schumacker, "A Characterization of Ten Hidden Surface Algorithms," *Comput. Surveys,* **6**(1), 1–55 (1974).
32. L. Carpenter, "The A-buffer: An Antialiased Hidden Surface Method," *Comput. Graphics* **18**(3), 103–108 (July 1984).
33. H. Gouraud, "Continuous Shading of Curved Surfaces," *IEEE Trans. Comput.* **C-20**(6), 623–629 (June 1971).
34. B. T. Phong, "Illumination for Computer Generated Pictures," *CACM* **18**(6), 311–317 (June 1975).
35. T. Whitted, "An Improved Illumination Model for Shaded Display," *CACM* **23**(6), 343–349 (June 1980).
36. F. C. Crow, "A Comparison of Antialiasing Techniques," *IEEE Comput. Graphics Appl.* **1**(1), 40–70 (Jan. 1981).
37. M. A. Z. Dippe and E. H. Wold, "Antialiasing Through Stochastic Sampling," *Comput. Graphics* **19**(3), 69–78 (July 1985).
38. M. E. Lee, R. A. Redner, and S. P. Uselton, "Statistically Optimized Sampling for Distributed Ray Tracing," *Comput. Graphics* **19**(3), 61–67 (July 1985).

SOFTWARE ELEMENTS

T. J. TARN
Washington University
St. Louis, Missouri

A. K. BEJCZY
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

INTRODUCTION

In recent years the use of industrial robots throughout industry has increased significantly. In many industrial applications such as painting, transporting, palletizing and many times even arc welding, programming of a sequence of movements is done through a "teaching by doing" process in which a human operator physically takes the robot hand through the desired sequence. When operating the robot, the controller reads the memory by a playback method. In such a situation, however, the robot can only repeat what it has been taught.

In some cases, where the kinematic design is simple enough and the dynamic control demands are low, on-board computing power is used to perform the necessary coordinate transformation between the joint coordinates, which are controlled directly, and the task coordinates, which are convenient to the task description. In such cases robot motions can be programmed from a computer keyboard, and the motions of robot arm joints can be controlled through the kinematic transformation of coordinates.

When the performance requirements for industrial robots are increased and involve accurate, fast, and versatile manipulations, dynamic effects become significant. This requires the control of a multi-input multi-output system described by a set of highly nonlinear, strongly coupled differential equations.

During the last decade, many papers discussed the control of robots through dynamic effects. Refs. 1–5 suggest using linearized system models as the basis for control. However, the implementation of a linear regulator in a robot system leads to many problems because the complex robot control has been synthesized based on the approximate linear model. This was demonstrated by a digital computer simulation of various control methods in Ref. 6.