Developing a Graphical Robotics Simulator

Chris Lattner and Ming-Shu Hsu

School of Engineering
University of Portland
Portland, OR 97229

ABSTRACT

An interactive robotics graphical computer simulation program, GRAS (Graphical Robot Animator and Simulator), has been developed. The GRAS program reads a data file of robot geometry data and displays the robot configuration as a 3D solid model. It simulates a teach pendant for user interaction. The program was written in Java language for maximum portability and application. GRAS is a generalized program so it can simulate a variety of robots with different configurations. Examples of PUMA560 and Pegasus robots are provided. The GRAS program is available on the web site *http://www.egr.up.edu/contrib/hsu/gras.1* and can be downloaded to the user's computer from the Internet.

INTRODUCTION

Today, many universities offer a robotics course. However, due to the complexity of the robot geometry the teaching of robotics has always been uneasy in the classroom. In addition, because of the high equipment and maintenance cost of the industrial robot many schools are teaching robotics without a real robot. To assist with this situation, an interactive graphical computer program, GRAS (Graphical Robot Animator and Simulator), has been developed. The use of this simulation program enables teaching robotics material in the classroom or in environments that are not conducive to purchasing and maintaining an industrial robot. This program, providing with a *simulated* hands-on experience, will be a great aid for teaching and learning the principles of robotics.

The GRAS program reads a data file of the robot geometry and displays the robot as a 3D solid model. It simulates a teach pendant for user interaction in two ways: Clicking the mouse on the specified joint will simulate pushing a button on the teach pendant of the joint and cause the robot to move the joint by one increment; Specifying all joints values will simulate executing a ***move*** command in a robot program and cause the robot to move from one location to another. The location of the end effector, and the configuration of the wrist, elbow, and shoulder joints are also calculated and displayed. GRAS is a generalized program so it can simulate a variety of robots with different configurations for up to six axes with any combination of prismatic and revolute joints. Examples of a six-axis Puma-560 robot and a five-axis Pegasus robot are included to illustrate the generality of the program. The program was written in Java language for maximum portability and application. It is available on the web site *http://www.egr.up.edu/contrib/hsu/gras.1* and can be downloaded to the user's computer from the Internet.

The development of the GRAS program is the result of an interdisciplinary research involving three fields, namely robotics, interactive computer graphics, and Internet Java programming. The following sections describe the theories in these three subjects and the implementation of the GRAS program.

THE ROBOT CONFIGURATION

An industrial robot is a series of links connected by joints moving in a 3D space. The relationships between the robot adjacent links can be described by the standard Denavit-Hartenberg (D-H) representation and the four parameters are link length *a*, twist angle α, offset *d*, and rotation angle θ [1]. Among these four parameters, three are fixed values representing the robot physical geometry and the other one varies allowing the robot joints movement. Depending on the type of joint, the joint variable is either the offset *d* for a prismatic pair or the rotation angle θ for a revolute pair. As depicted in Figure 1, the four parameters between any two adjacent links are defined as follows:

a_i is the link length from the intersection of the z_{i-1} axis with the x_i axis to the origin of the i^{th} coordinate system along the x_i axis.

α_i is the twist angle from the z_{i-1} axis to the z_i axis about the x_i axis.

d_i is the offset distance from the origin of the $(i\text{-}1)^{th}$ coordinate system to the intersection of the z_{i-1} axis with the x_i axis along the z_{i-1} axis.

θ_i is the joint angle measured from the x_{i-1} axis to the x_i axis about the z_{i-1} axis.

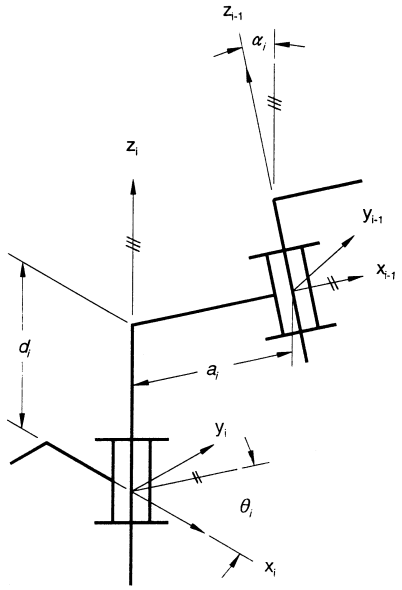

Figure 1: Link Parameters

Since the robot's joints move independently, we need to have independent coordinate systems to describe the relationships between any two adjacent links. Assigning an orthonormal Cartesian coordinate system to each link at its joint axis and one at the base, we can establish the kinematic configuration of each link[2,3]. Then, by applying the Denavit-Hartenberg representation to each link, a homogeneous transformation matrix ${}^{i-1}\boldsymbol{A}_i$, known as the $\boldsymbol{A}$ matrix, relating the i^{th} coordinate frame to the adjacent $(i\text{-}1)^{th}$ coordinate frame can be derived. The $\boldsymbol{A}$ matrix for a revolute joint is given below.

$$ {}^{i-1}\boldsymbol{A}_i = \boldsymbol{T}_{z,d}\,\boldsymbol{R}_{z,\theta}\,\boldsymbol{T}_{x,a}\,\boldsymbol{R}_{x,\alpha} = \begin{vmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{vmatrix} $$

where

$\boldsymbol{T}_{u,v}$ = a transformation matrix of translation along u-axis by a distance v

and $\boldsymbol{R}_{m,n}$ = a transformation matrix of rotation about m-axis by an angle n

The ${}^{i-1}\boldsymbol{A}_i$ matrix transforms a point in the i^{th} coordinate frame to the adjacent $(i\text{-}1)^{th}$ coordinate frame. Thus, through successive transformations of the $\boldsymbol{A}$ matrix, the location of a point on the link i can be transformed to the base coordinate system.

$$ \{{}^{o}\mathbf{P}\} = {}^{o}\boldsymbol{A}_1\,{}^{1}\boldsymbol{A}_2\,\ldots\,{}^{i-2}\boldsymbol{A}_{i-1}\,{}^{i-1}\boldsymbol{A}_i\,\{{}^{i}\mathbf{P}\} $$

where

$\{^{i}\mathbf{P}\}$ = the position vector of the point **P** expressed in the i^{th} coordinate frame

$\{^{o}\mathbf{P}\}$ = the position vector of the point **P** expressed in the base coordinate frame

RENDERING INTERACTIVE COMPUTER GRAPHICS

The graphics rendering subsystem of GRAS is very important – without it, no visual feedback would be possible. Because of the importance that it plays in the user's understanding of the robot model, it is imperative that the rendition be an accurate model of the current robot geometry. Also important to the GRAS experience is the speed of the rendering engine; high performance is required for interactive graphics. Although a photo-realistic model of the robot geometry could be produced, this would be unsuitable for real-time animation. Many user interface factors, such as these, have influenced the design of the rendering engine built into GRAS.

The first stage of the rendering pipeline converts the local coordinates of each robot link into a set ***W*** of vertices in the world coordinate system. The current joint angles of the model and the link geometry are used to calculate the resulting world geometry for all of the vertices (as described in the robot configuration section); this geometry information forms the set ***W***. The next step of the rendering pipeline converts the set ***W*** into a set ***E*** of vertices in the eye coordinate system by rotating and translating the world coordinates into the view space defined by the current location of the eye. The final step of the view transformation converts from the eye coordinate system to two dimensional screen coordinates system, forming a new set of vertices ***S*** that are suitable for drawing in the current display window. See Figure 2 below.

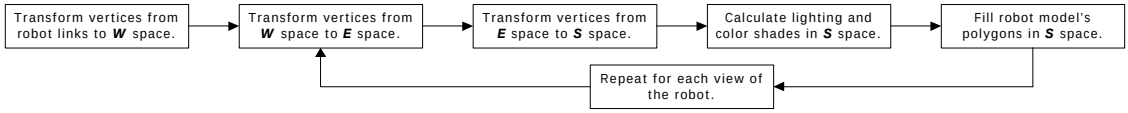

Figure 2: GRAS Rendering Pipeline

Internally, the geometry information for each individual link of the robot is kept apart from the other links. When a redraw of the robot is requested, the current joint angles are used to compile a single model of the robot with the joints fixed into the position to be drawn. Every surface in the resulting model contains a list of vertices (which are connected to form polygons), and a color to shade the surface. Each vertex is represented as a homogenous column matrix of the form $\mathbf{v} = |X \quad Y \quad Z \quad 1|^{T}$. Thus, each vertex in the surfaces is represented in three dimensions with an X, Y, and Z component.

Once the geometry of the robot has been compiled into ***W***, each view of the robot transforms the world coordinates of the vertices into viewing coordinates, based on the location and orientation of the eye. To transform vertices from ***W*** to ***E***, a transformation matrix $\mathbf{T}_v$ is used. This transformation matrix can accommodate translation (X_t, Y_t, Z_t) along the X, Y, and Z axes as well as rotation about the X and Y axes. The transformation matrix $\mathbf{T}_v$ may be represented as the composition of separate matrices to rotate about the X-axis ($\mathbf{X}_{x,\theta}$), rotate about the Y-axis ($\mathbf{R}_{y,\varphi}$), and translate along the x, y, and z axes ($\boldsymbol{\tau}_f$).

$$\mathbf{R}_{x,\mathrm{q}} = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\mathrm{q} & -\sin\mathrm{q} & 0 \\ 0 & \sin\mathrm{q} & \cos\mathrm{q} & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \qquad \mathbf{R}_{y,\mathrm{j}} = \begin{vmatrix} \cos\mathrm{j} & 0 & -\sin\mathrm{j} & 0 \\ 0 & 1 & 0 & 0 \\ \sin\mathrm{j} & 0 & \cos\mathrm{j} & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \qquad \boldsymbol{\tau}_f = \begin{vmatrix} 1 & 0 & 0 & X_t \\ 0 & 1 & 0 & Y_t \\ 0 & 0 & 1 & Z_t \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

$$\mathbf{T} = \mathbf{R}_{x,\theta} \cdot \mathbf{R}_{y,\phi} \cdot \boldsymbol{\tau}_f = \begin{vmatrix} \cos\phi & 0 & -\sin\phi & X_t \\ -\sin\phi\ \sin\theta & \cos\theta & -\cos\phi\ \sin\theta & Y_t \\ \sin\phi\ \cos\theta & \sin\theta & \cos\phi\ \cos\theta & Z_t \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

When the user interacts with a view of the robot, a new $\mathbf{T}_v$ matrix is calculated based on the user request. If the user requests that the model be rotated 10° about the Y-axis, a new $\mathbf{T}_v$ matrix is created with the correct values for a 10° rotation. This new $\mathbf{T}_v$ matrix is then composed with the original $\mathbf{T}_v$ viewing matrix to update the camera transformation and the view is redrawn with the new $\mathbf{T}_v$.

To calculate the points in the ***E*** set from points in the ***W*** set, $\mathbf{T}_v$ is pre-multiplied into each vertex in ***W***. The resulting vertices form the ***E*** set. Thus,

$$\mathbf{V}_{\boldsymbol{E}} = \mathbf{T}_v \cdot \mathbf{V}_{\boldsymbol{W}}$$
$$\begin{vmatrix} X_{\boldsymbol{E}} & Y_{\boldsymbol{E}} & Z_{\boldsymbol{E}} & 1 \end{vmatrix}^T = \mathbf{T}_v \cdot \begin{vmatrix} X_{\boldsymbol{W}} & Y_{\boldsymbol{W}} & Z_{\boldsymbol{W}} & 1 \end{vmatrix}^T$$

To calculate points in the set ***S***, the vertices in ***E*** are multiplied by a projection matrix **P** to project the vertices into the Z=1 plane, and to scale the X and Y components to fit the current display window. For this transformation, **P** is defined as follows:

$$\mathbf{P} = \begin{vmatrix} \frac{Z_F \cdot W}{Z} & 0 & 0 & W \\ 0 & \frac{Z_F \cdot H}{Z} & 0 & H \\ 0 & 0 & \frac{1}{Z} & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \qquad V_{\boldsymbol{S}} = \mathbf{P}\mathbf{V}_{\boldsymbol{E}} = \begin{vmatrix} \frac{Z_F \cdot W}{Z} & 0 & 0 & W \\ 0 & \frac{-Z_F \cdot H}{Z} & 0 & H \\ 0 & 0 & \frac{1}{Z} & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \cdot \begin{vmatrix} X \\ Y \\ Z \\ 1 \end{vmatrix} = \begin{vmatrix} \frac{X \cdot Z_F \cdot W}{Z} + W \\ \frac{-Y \cdot Z_F \cdot H}{Z} + H \\ 1 \\ 1 \end{vmatrix}$$

Where W is one half of the target window's width, H is one half the target window's height, Z_F is the current Zoom-Factor, and X, Y, Z are the current point to be projected. After this transformation, the X and Y components of the resulting vertex are in display window coordinates, and the Z component of the resulting vector is 1. As such, this is a two dimensional point ready to be plotted in the client window coordinates.

As an example, if the current display window is of size 640x480, W=320 and H=240. Assuming the Zoom Factor is set to 1, projecting point $(-12, 17, 22)^T$ returns a value of $(145, 54, 1, 1)^T$. To plot this vertex, the pixel at (X=145, Y=54) is filled. To draw a polygon, multiple vertices are projected and lines are drawn between them – these lines are then filled with a solid color.

To shade the polygons in the model, Lambert's reflection model is used. Lambert's model states that the intensity of the light reflected off of a surface is determined by the orientation of the surface with respect to the light source. Specifically, this intensity S_i is proportional to the cosine of the angle between the surface normal and the light source vector.

$$S_i = \cos\theta_{NL} = \frac{\mathbf{N}_x\mathbf{L}_x + \mathbf{N}_y\mathbf{L}_y + \mathbf{N}_z\mathbf{L}_z}{|\mathbf{L}| \cdot |\mathbf{N}|}$$

Where **N** is the surface normal vector and **L** is the light source vector.

In practice, a number of factors simplify the equation above. First, because the light is fixed in location, it has a pre-normalized length of 1.0. Also, because the light vector is directed along the Z-axis, the $\mathbf{L}_x$ and $\mathbf{L}_y$ portions of the vector are always zero (L_z is always 1.0, but shown for context). This considerably

simplifies the above equation, leaving the equation $S_i = \cos\theta_{NL} = \mathbf{N}_z\mathbf{L}_z / |\mathbf{N}|$. The evaluation speed of this formula is crucial to GRAS, because its simplicity is directly related to GRAS' performance.

Once the surface intensity S_i has been calculated, the polygon corresponding to the surface is filled with an adjusted color based on the color associated with the surface. Each surface maintains a base color, which is the color of the surface under 100% light intensity. This color is represented as a composition of red, green and blue components (*R*, *G*, *B*) which have a range from 0 to 1. The resulting color is calculated by scaling each component by the surface intensity. This yields the surface color S_c: $S_c = (S_i \cdot R, S_i \cdot G, S_i \cdot B)$. Vertices from the set ***S*** are used to find the coordinates for the two-dimensional vertices of the current surface. This surface is then rendered by drawing a solid polygon between the transformed points in screen space with the color S_c.

THE INTERNET AND THE JAVA PROGRAMMING LANGUAGE

For many reasons, the Internet has rapidly become a very important way of gathering and dispersing information. The Java programming language is uniquely suited for Internet programming tasks, primarily because of its wide spread availability and common use on the Internet. For exactly these reasons, we chose the Internet as the distribution media for GRAS and Java as the programming language. This section examines the key implications of the Internet and Java on GRAS.

The Internet is an ideal distribution media for GRAS because it allows free access to many potentially interested people, and allows the software to be run directly from a browser. As a system of cross-links, the Internet allows web pages with mutual themes to be linked together… allowing one to find related information quickly. This specific property allows web pages that focus on robotics topics to link to the GRAS page. This structure builds a community of robotics related pages, and GRAS belongs to that community.

Another key factor that influenced our decision to support the Internet is the ease of use that it provides. This allows people who are fluent in robotics techniques and interested in the teaching applications of GRAS, to access and use GRAS - even if they are not particularly computer literate. As more people are exposed to the Internet, GRAS is accessible to a potentially larger audience. The Internet is truly the distribution medium of the future.

The Java language is a relatively new programming language developed by Sun Microsystems. Java features an elegant programming style and sophisticated integration with the Internet. Most importantly, however, Java aims to provide truly cross-platform binary executables. This means that the same executable file can run on a PC, a Macintosh, or a Unix workstation – without any additional support effort by system administrators. This cross platform architecture is key for GRAS, because many potential users are not familiar with the Unix environment that many robotics tools require.

The elegant programming style that is encouraged by the Java language enables the development of very modular and extensible programs. In particular, it will be easy to extend GRAS to provide new rendering modes, trajectory motion models, and robot geometry solvers. This allows GRAS to be easily extended in the future to simulate and test new algorithms in the robotics realm, as well as to adapt for potential class assignments.

Overall, the Java language allows GRAS to be a more useful and friendly program. By being a cross platform application, users of many different operating systems and applications can run GRAS on the machines available to them. This also allows heterogeneous computing environments to maintain one copy of the binary executable for all of their systems. In addition to compatibility, the extensibility of the Java language enables new features to be added very easily, allowing GRAS to be extended into an even more useful tool. This is crucial for using GRAS in a teaching environment where the tool must be tailored to the student's changing levels of understandings.

The seamless integration of Java with web pages allows dynamic delivery of GRAS program, and direct execution from the browser. This allows users to be able to transfer their knowledge about navigating web pages into a way to start the execution of a program. The flexible nature of the Internet even allows the user to download the entire GRAS program to their local machine, which allow it to be run on machines that are not connected to the Internet at all.

EXAMPLE OF THE GRAS PROGRAM

Two robots, Puma-560 and Pegasus, were chosen as the examples for illustrating the implementation of the GRAS program. When the program was executed, the robot geometry was displayed along with two menus: Teach Pendant and Robot Inspector, as shown in Figures 3 and 4 for the Puma-560 robot. The Teach Pendant menu serves as an input device and the Robot Inspector menu is an area providing the robot geometry information.

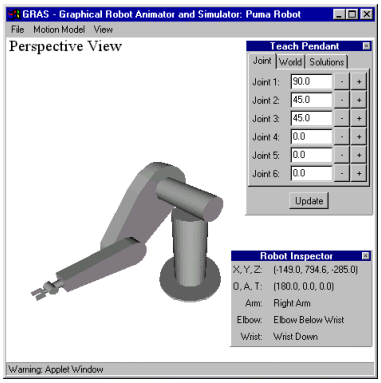

Figure 3: Puma Robot

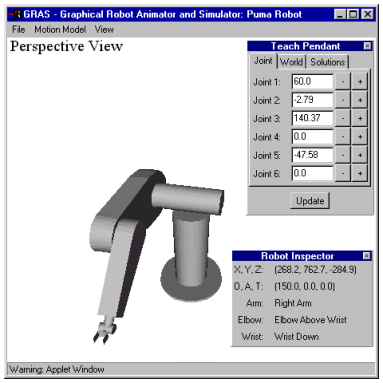

Figure 4: Puma Robot

There are two modes in the Teach Pendant menu: Joint and World. When the Joint mode was selected, the six input values represent the values to be applied to the six joints respectively. When the World mode was selected, the first three input values represent the (X, Y, Z) coordinates of the end effector position while the next three values represent the rotation angles (O, A, T) of the end effector orientation.

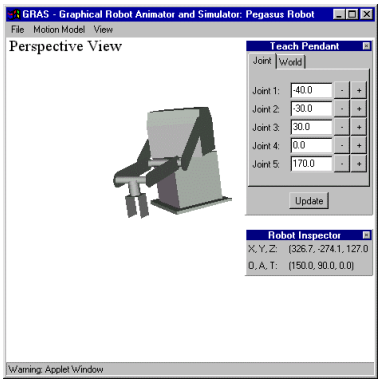

Figure 5: Pegasus Robot

To illustrate the generality of the program, a five-axis Pegasus robot was simulated in Figure 5. By only providing a geometry file to describe the robot, the GRAS program is able to simulate a robot of any complexity, with any geometry configuration. It is also capable of performing direct kinematics transformations on these arbitrary robots.

CONCLUSION

An interactive graphical computer simulation program, GRAS, has been developed. The program simulates controlling a robot arm with a teach pendant, displaying the robot with a 3D solid model. GRAS is available from the Internet *http://www.egr.up.edu/contrib/hsu/gras.1*, where it can be downloaded to the user's computer for use. Although the program presented in this paper is complete for these purposes, the authors would appreciate feedback on suggested improvements.

REFERENCES

1. J. Denavit and R.S. Hartenberg, "*A Kinematic Notation for Lower Pair Mechanisms Based on Matrices*," J. Appl. Mech., 1955.

2. K.S. Fu, R.C. Gonzalez, and C.S.G. Lee, *Robotics: Control, Sensing, Vision, and Intelligence*, McGraw-Hill Book Company, 1987.

3. Richard P. Paul, *Robot Manipulators: Mathematics, Programming, and Control*, The MIT Press, 1982.

4. Chan S. Park, *Interactive Microcomputer Graphics*, Addison-Wesley Publishing Company, 1985.

5. Foley, Van Dam, Feiner, and Hughes, *Computer Graphics: Principles and Practice*, Addison Wesley, 1993.