Appendix A

The Peabody Object Representation

Peabody is a representation for complex articulated geometric objects. It represents *figures* composed of *segments* connected by *joints*, also under the influence of *constraints*.

Jack is an interactive system for constructing and manipulating peabody objects. It is crucial to understand peabody before using *Jack*, since so many of the features of *Jack* deal intrinsically with peabody objects and rely heavily on the syntax of the peabody language. You may design **peabody** environments without using *Jack* at all, but *Jack* was designed to make this task easier.

A.1 Introduction to Peabody

A peabody environment consists of a number of individual *figures*, each of which is a collection of *segments*. The segments are the basic building blocks of the environment. Each segment has a "geometry." It represents a single physical object or "part", which has shape and mass but no movable components. The geometry of each segment is represented by a *psurf*, which is generally a polyhedron or a polygonal mesh.

The term "figure" applies not only to articulated, jointed figures such as a human body: any single "object" is a figure. It need not have moving parts. A figure may have only a single segment, such as a coffee cup, or it may be composed of several segments connected by *joints,* such as a robot. The term "object" is used here very loosely; it has no special significance. We use it only to denote some part of the peabody environment.

The term *psurf* refers only to the representation for the geometry of a segment, which is the graph of nodes and edges typically drawn as the wireframe or shaded image of the segment. In the case of a figure with a single segment, it is sometimes convenient to refer to it as a "psurf", but that is not technically correct.

Joints connect segments at attachment points called *sites.* A site is a local coordinate frame specified relative to the coordinate frame of its segment. Each segment may have several sites. Joints connect sites on different segments within the same figure. Sites need not lie on the surface of a segment, but generally they will. A site is a coordinate frame which has an orientation as well as a location.

A.1.1 The Connectivity of Peabody Objects

The peabody environment can be visualized as a directed graph. The segments are the nodes of graph, and the joints form the edges. Figures are maximal subgraphs spanned by joints. It is important to remember that joints do not connect segments directly. Joints connect segments through sites.

A.1.1.1 The Hierarchy of Peabody Objects

Peabody avoids representing objects with a strict hierarchy by encouraging you to think of figures as collections of segments and joints. However, there *is* an underlying hierarchy, and as in the case of moving a figure or adjusting a joint, it is important to remember what the hierarchy is because it has an effect on which objects remain fixed and which move when the joints change angles.

In order for the location of an articulated figure to be well defined, we must designate some element of the figure as its origin. The location of a peabody figure is specified through a site designated as the *root*. The root site roughly corresponds to the "origin" of the figure, and it provides a handle by which the location of the figure may be specified. Viewing the figure as a tree, the root of the figure is the root of the tree. The root site of a figure may be changed interactively in *Jack* from time to time depending upon how you want to manipulate the figure. This is important because the root of the figure serves as the origin of rotation and translation when you manipulate a figure interactively.

It is not allowable to define figures with closed loops of joints in peabody, although the syntax of the peabody language does not prevent you from doing so.

A.1.1.2 A Metaphor for the Connectivity of Peabody Objects

To visualize the graph of a peabody environment, imagine a collection of simple objects, such as machine parts, floating around in zero-gravity space. Several of these objects are connected to each other with hinges. The objects are the segments and the hinges are the joints. The placement of the hinges on the objects is described by the placement of the sites on the segments. A collection of segments hinged together form a figure. There may be several figures floating around. Some figures may consist of lots of segments and hinges. Other figures may have only a single segment. No segment is part of a figure unless it is hinged to the rest of the figure's segments. You need not think of the segments and joints in the figure has having a strict hierarchy. The joints connect segments in completely arbitrary ways.

Each figure is nailed in place in space through its root site. The global placement of this field in space defines in turn the placement of all other segments in the figure. When you rotate a figure interactively, it rotates around its root site. When you translate a figure, it may move along the global coordinate axes of this frame. *Jack* allows you to interactively change the root, so if it becomes convenient to nail the figure down in a different way, it is possible to do so.

When you bend a hinge at one of the joints of a figure, the segments on one side of the joint will remain fixed, and the ones on the other side will move. Which side moves and which remains fixed depends on how the figure is rooted: the ones on the rooted site remain fixed.

We described the graph of the environment as *directed*, and the joints have a distinct direction. However, the directed-ness of a joint does not affect which segments stay fixed and which move as the joint angles change. The direction defines only the order in which the rotations of a joint are applied to produce the complete transformation between the sites which the joint connects. The details of joint definitions and degrees of freedom will be discussed in Section A.3.5.1.

A.1.2 The Geometry of Peabody Objects

It is very important to understand how the geometry is associated with a peabody figure. Each segment may have a psurf associated with it, but the shape of the psurf itself does not affect the topology of the figure. In *Jack*, it serves only as the *image* for the segment. The underlying topology of the figure, in terms of the "lengths" of the segments and the placement of the joints, is defined by the site locations relative to each segment, and it is completely independent of the psurf geometry. Sometimes sites will lie on the surface of the psurf, but this is by design rather than requirement.

There is no enforced relationship between the geometry and the length of a segment. In fact, segments don't really have a "length." If you define the length of a segment as the distance between the joints at either end, then this is only well defined when a segment has only two joints. But peabody allows segments to have any number of joints.

The geometry of a psurf is specified relative to the coordinate origin of the segment. This means that the (x, y, z) coordinates of the vertices of the polygons of the psurf are interpreted and drawn relative to the axes of the coordinate frame of the segment, *not* the world coordinate frame. Each psurf is designed in its own coordinate system.

Take a moment to consider what (x, y, z) cartesian coordinates really mean. These coordinates are only meaningful when interpreted in the context of a coordinate frame. Frequently, the coordinate frame is implicitly the world coordinate frame, but this not necessarily true. The (x, y, z) coordinates of a point define the location of a point relative to the coordinate frame by specifying displacements from the origin of the frame along the x, y, and z axes of the frame, respectively: start at the origin of the frame, travel x units down the x axis, then travel y units along the y axis, then travel z units along the z axis.

As an example, consider the two cubes shown in Figures A.1 and A.2. Along with each picture is the syntax for the psurf which defines the geometry. The details of the syntax for the psurf files is described in Section A.4. These two objects have the same "shape" but their origins are different. In Figure A.1, the origin is at the one of the corners. In Figure A.2, the origin is in the middle of the cube. It is important to realize that this difference is in the *nodes* of the psurf, not in the definition of the location of the figure itself. The origin of the psurf is implicit: it is *with respect to this frame* that the polygons of the object are interpreted. Peabody gives you great freedom in how psurfs are designed, since the sites may be place anywhere on the segment. This gives you the ability to attach joints to segments in various places.

```
0.00    0.00    0.00
0.00    100.00  0.00
100.00  100.00  0.00
100.00  0.00    0.00
0.00    0.00    100.00
0.00    100.00  100.00
100.00  100.00  100.00
100.00  0.00    100.00
;;
1 2 3 4;
1 4 8 5;
3 7 8 4;
1 5 6 2;
5 8 7 6;
2 6 7 3;
;;
```

Figure A.1: `A Cube`

```
-50.00  -50.00  -50.00
-50.00  50.00   -50.00
50.00   50.00   -50.00
50.00   -50.00  -50.00
-50.00  -50.00  50.00
-50.00  50.00   50.00
50.00   50.00   50.00
50.00   -50.00  50.00
;;
1 2 3 4;
1 4 8 5;
3 7 8 4;
1 5 6 2;
5 8 7 6;
2 6 7 3;
;;
```

Figure A.2: `Another Cube`

This can sometimes be very confusing because it is the *sites* on the segment, and the joints which connect them, which define the location of the segment relative to other segments in the figure. Many times, it is convenient to define a site which lies at the coordinate origin of a segment. This is particularly true of an elongated segment like an arm, which has a distinct proximal and distal end. In this case, the proximal end may lie at the coordinate origin of the segment, and the distal end may lie down the z axis, for example. In this case, it may seem that the geometry is specified relative to the proximal end, but remember that the geometry is relative to the origin of the segment. The proximal site just happens to lie at that origin.

Drawing a diagram may solve lots of confusion. It is usually a good idea to draw a diagram of the segment with an explicitly labeled origin. Then draw the sites, such as proximal and distal, *away* from the origin, even if in fact they are coincident. This will help to reinforce the fact that they are specified relative to the origin of the segment.

A.1.2.1 A Metaphor for the Construction of Peabody Figures

When designing the geometry of the segments of an articulated figure, it is best to proceed logically in a manner similar to what we might do if we were constructing a hinged mechanism out of wood, metal, or plastic. The most logical thing to do is to design each part one at a time. We begin by choosing an origin for the part and crafting its shape in terms of dimensions measured from that origin. After designing all of the individual moving parts, we proceed to drill holes in each part into which to bolt the hinges. Where do the holes go? We again calculate the location of the holes in terms of displacements from the origin of the segment. Next, we bolt the hinges into the holes and bend them to the correct angles (Let's assume that the hinges are stiff enough hold the figure in place). Finally, we choose a special point on the figure and place this point at the proper place so that the contraption is located in the desired position on the table, floor, or wall.

In the translation of this metaphor, the parts are the segments and the hinges are the joints. The shape of the part is defined in terms of a psurf, whose coordinates are interpreted relative to the coordinate origin of the segment. This coordinate origin is not relative to anything: other things are relative to it! The holes for the hinges correspond to the sites: they are measured from the coordinate origin of the segment. The special point on the contraption through which we fix it to the wall or floor corresponds to the figure's root site.

A.2 The Mechanics of the Peabody Language

The syntax of the peabody language somewhat resembles a programming language, except that it defines static elements, not actions.

A.2.1 Arithmetic Expressions

The peabody language employs a powerful arithmetic expression parser and symbol table, so that any part of the language which requires a numerical value accepts a general arithmetic expression. The syntax of the expressions is similar to an ordinary programming language. The operators and their precedence are shown in Figure A.3. The operators at the top have the greatest precedence.

()	parentheses
-	unary minus
^	exponentiation
/	division
*	multiplication
-	subtraction
+	addition

Figure A.3: The Precedence of Peabody Arithmetic Operators

Variables need not be declared before use: their type is determined by context, although a warning message will be issued if a variable is used before being assigned a value.

A.2.2 Units

Many numerical values in the peabody language refer to physical measurements: angle, distance, mass. Peabody allows such values to be quantified by their type, so that values may be entered in any particular units. Whenever a value is specified, its units should also be specified so there is no confusion about which units are being used. This convention will save much confusion as files are written as used at later dates.

For angles, the legal types are radians and degrees, using the keywords `rad` and `deg`. The default is degrees. For distances, the legal types are millimeters (`mm`), centimeters (`cm`), meters (`m`), inches (`in`), feet (`ft`), and yards (`yd`). The default is centimeters.

A.2.3 Homogeneous Transformations

Peabody relies heavily on the specification of homogeneous transformations, and the peabody language has a rather simple mechanism for describing such transformations. Transformations may be expressed as a sequence of simpler, primitive transformations such as rotation and translation. The translation operator is `trans`, and it takes three arguments, giving the translation in x, y, and z. Rotation may be described with the `xyz` operator, which specifies rotation in terms of angles around the local x, y, and z axes, in that order. For example,

```
t = xyz(10deg,20deg,30deg);
```

specifies a rotation transformation which is formed by a rotation of 10° around the x axis, followed by a rotation of 20° around the *rotated* y axis, followed by a rotation of 30° around the *rotated* z axis. This operator can be used for simple rotations around a single coordinate axis by using zeros for two of the angles.

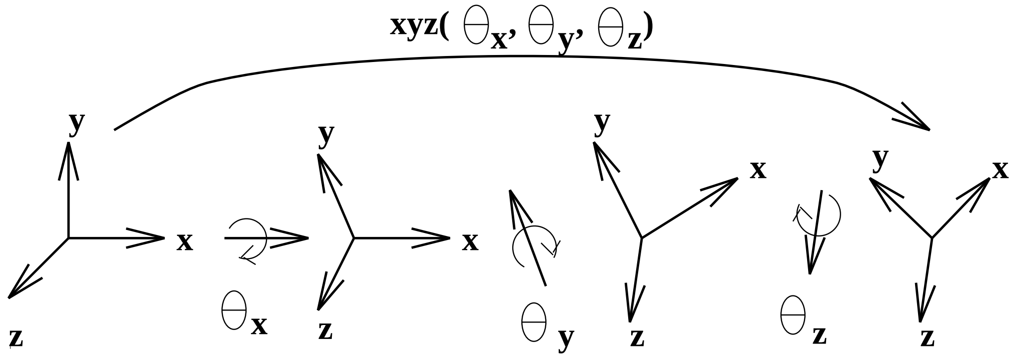

Figure A.4: The `xyz` Rotation Operator

A.2.3.1 Multiplying Homogeneous Transformations

The peabody language represents homogeneous transforms as 4×4 matrices in the form:

$$\begin{bmatrix} x_0 & x_1 & x_2 & 0 \\ y_0 & y_1 & y_2 & 0 \\ z_0 & z_1 & z_2 & 0 \\ p_0 & p_1 & p_2 & 1 \end{bmatrix}$$

The translation vector (p_0, p_1, p_2) is in the bottom row of the matrix.

When transformations are multiplied, the product may be interpreted in two ways. The most intuitive way corresponds to local transformations when applied *right* to *left*. For example, the transformation given by

```
t = xyz(90deg,0,0) * trans(0,100cm,0);
```

may be interpreted as first a translation of 100 cm along the y axis, followed by a rotation of 90° around the *translated* x axis. The ordering is critical, since transformations don't commute.

Alternatively, the product of transformations may be interpreted in global coordinates when read *left* to *right*. The above transformation may be interpreted as a rotation of 90° around the x axis, followed by a translation of 100 centimeters along the *original* y axis.

When *Jack* writes a transformation, it writes the rotation part followed by the translation part, expressed in terms of the `xyz` operator, no matter how the expression was originally specified. This product can be interpreted as a rotation followed by a translation with respect to the base coordinate frame, or alternatively a translation followed by a rotation around the *translated* axes.

The exponentiation operator may be applied to transforms, with the usual meaning. Raising a transform to the power of -1 yields the inverse of the transform.

A.2.3.2 Manipulating Transformations Interactively in Jack

You can experiment interactively with these transformations in *Jack* by moving a figure. As you move the figure, the peabody description of the figure's root appears at the bottom of the window. You can see the description of the transformation as it is being manipulated.

Another valuable exercise is to enter the transformations from the keyboard by hitting `^K` while moving the figure. You will be prompted in the message window to enter the transformation. Experiment with several transformations to get a feel for how this syntax works, particularly in terms of the ordering of the transformations. The details of how to enter values from the keyboard in *Jack* are described in Section 5.8.

A.2.4 Names and Identifiers

All constructs in the peabody language have names, and each name is local to the construct to which it belongs. This ambiguity may be resolved by prefixing the name of construct with the name of its parent, separated by a period. For example, each figure in the environment has a distinct name, and each segment within each figure has a name which is unique only within that figure. Therefore, two figures, say `fred` and `ethel`, may have segments named `head`. The identifier `head` does not uniquely specify *which* head, so the use of `head` must be prefixed with either `fred` or `ethel`, as in `fred.head` or `ethel.head`.

Likewise, joints are local to the figures to which they belong, and sites are local to the segment to which they belong. In the case of sites, a double prefix must be used to uniquely specify a site in the environment. This follows a general rule of thumb that when constructs are named, they must be properly qualified given the context. When naming segments within a figure, the figure name is usually understood and may be omitted.

One exception to the above rule is the pseudo-segment `world`, which is the base coordinate frame for the environment. The world is not a part of any figure, so a reference to one of its sites has only one prefix, as in `world.base`.

Names of constructs and variable identifiers must consist of upper and lower case alphabetic characters, digits, and underscores, and must not begin with a digit. It is important that names of constructs in peabody do not conflict with keywords. Since most keywords are frequently used terms, this rule must be carefully considered when constructing peabody figures. The peabody keywords are listed in Table A.1

It is important to remember the difference between identifiers and strings, especially in places in the peabody language where file names are required. All strings, such as file names, must be enclosed in double quotes. Otherwise, they would be interpreted as variable references.

A.3 Peabody Construct Declarations

The term "construct" refers to any of the basic elements of the peabody object representation: figures, segments, sites, joints, attributes, and lights. The peabody language consists primarily of assignment statements which define the properties of the peabody constructs. Each assignment must be qualified with the construct to which it applies. This is generally done by grouping the assignments together into *blocks* which set forth the construct name and type, similar to a data structure declaration in a programming language. The elements within the block are then automatically associated with that construct.