Linear Geometry in E3

Objects

Points
Displacements
Planes
Lines
Coordinate Systems
Bounded Geometry

Algorithms

Interrogations
Transformations
Interrogations on Bounded Objects

Points

A **point** is an abstraction of a place (or position) in E3.

Representation

Three coordinates with respect to some coordinate system

Displacements

A **displacement** is an abstraction of a movement from one place to another.

Representation

Three components with respect to some coordinate system

Arithmetic of Points and Displacements

Cannot add points: Can add displacements

Can subtract points to give a displacement

Can add a displacement to a point to give a new point

Cannot scale points: Can scale displacements

Can take a weighted mean of points to give a new point

Can multiply a vector of points by a matrix only if each row of the matrix sums to

unity - the corresponding result is a point

zero - the corresponding result is a displacement

Planes

A plane is a particular type of bivariate Point Set. It has uncountably many points and so it needs a finite representation. Sets have three types of representation, of which two are suitable for infinite sets.

Representations

Implicit: $f(x, y, z) = 0$

$f(P) = [P - Q].N = 0$

Parametric: $P(u, v)$

Generative map from $I\!R^2$.

$P(u, v) = Q + uU + vV$

$P(u, v, w) = wQ + uR + vS, \; u + v + w = 1$

Note that the implicit representation has six numbers (one point and one displacement), while the parametric has nine (one point and two displacements, or three points). We call the number of numbers in a representation the **number of degrees of freedom**. If two representations of the same kind of object have different numbers of degrees of freedom, the representation with more must be **redundant**, and either there are limitations on the values the numbers can take, or else different combinations of values can represent the same object.

In this case both representations are redundant, since a minimal representation needs only three numbers.

However, such a representation is rather inconvenient in this case, since planes form a projective space rather than a Euclidean one, and so we accept the redundancy.

It is always bad practice to insist on values in the representation satisfying some equality limitations, because numerical rounding is always likely to break those rules. We can, however, express such limitations as recommendations, in which case a representation satisfying them is called **canonical**.

In this case it is useful for the displacement N to have a unit magnitude, because then $f(P)$ gives the value of the signed distance of P from the plane.

We can always compute the less redundant form from a more redundant one. If we try to compute the more redundant form from the less, then arbitrary choices have to be made.

Suitable canonical rules are that U and V should be mutually perpendicular unit displacements, but the choice of the angle at which U lies in the plane is totally arbitrary.

Lines

A line is a particular type of univariate Point Set.

Representations

Parametric: $P(t)$ Generative map from $I\!R$.

$P(t) = Q + tT$

$P(t) = sQ + tR, s + t = 1$

There are also two implicit forms, one based on the distance of a query point from the line, the other on Plucker coordinates. The distance form is numerically unsatisfactory, and the Plucker coordinates are beyond the scope of this course.

Coordinate Systems

This is actually a map ($I\!R^3$ to $I\!R^3$), and what we store is the set of coefficients for an algorithm which implements the mapping.

Because of the importance of solid body transformations and the translation subset, we focus almost exclusively on **rectangular Cartesian coordinate systems**.

Representations

[4x4 matrix]

Origin point and Axis displacements

[Translation displacement and Rotation quaternion]

The same data can represent both the map and its inverse. Conversion from objects stored with respect to the coordinate system to their representation with respect to the implicit system is called **mapping out**: conversion from implicit coordinates to local coordinates is called **mapping in**.

Interrogations

There is a nice clean set of primitive **incidence constructions** on points, lines and planes, simple to implement when all the operands are represented with respect to the same coordinate system:

point x point $\rightarrow$ line
point x point x point $\rightarrow$ plane
plane x plane $\rightarrow$ line
plane x plane x plane $\rightarrow$ point
point x line $\rightarrow$ plane
plane x line $\rightarrow$ point

This is not an irreducible set, because it is possible to define some in terms of others.

All of these can have singular inputs, for which the output is undefined. Sometimes there may be no output (e.g. intersection of two parallel planes), and sometimes there may be multiple solutions(e.g. intersection of two coincident planes).

Such singularity typically appears in numerical linear geometry as a divide by zero somewhere in the algorithm.

When the input configuration is almost but not quite singular, the computation will be prone to rounding errors. These can be avoided to some extent by use of rational numbers rather than floating point, and the use of indefinite precision (multi-length integers for numerator and denominator), but these are expensive and not used in most commercial software.

There are also **incidence tests**:

point x point $\rightarrow$ true/false
point x line $\rightarrow$ true/false
point x plane $\rightarrow$ true/false
line x plane $\rightarrow$ true/false
line x line $\rightarrow$ true/false
plane x plane $\rightarrow$ true/false

but these are always numerically fraught.

There are also primitive **metric computations**, on points, lines, planes, and distances, easy to implement when the geometric objects are represented with respect to a Cartesian coordinate system:

point x point $\rightarrow$ distance
point x plane $\rightarrow$ distance
line x line $\rightarrow$ distance

The second of these gives a signed distance, which is positive if the point lies on one side of the plane and negative if it lies on the other side. The third also gives a signed distance, positive if the first line twists clockwise about the second.

These are both examples of the use of **orientation**[2].

In Cartesian geometry, we can also calculate the common perpendicular of two lines.

line x line $\rightarrow$ line

Transformations

These correspond to changes of coordinate system, and the best way of defining a transformation is to specify a coordinate system which is the image of the implicit coordinate system under the required transformation, and then map the points out of that system back into the implicit system.

Transforming objects other than explicit point sets may involve non-trivial algorithms, especially under the most general transforms. For example, transforming a plane by just a translation when the before and after coordinate systems are spherical polar gives a good argument for using Cartesian systems.

[2]

Mapping Out

The process of mapping out a point is given straightforwardly by the defining equation

$$P := O + xX + yY + zZ$$

where O is the origin of the auxiliary coordinate system, and X, Y and Z are its unit axis vectors (displacements), and x, y and z are the coordinates held with respect to the auxiliary coordinate system.

This can be expressed rather more elegantly as

$$P_I := O_I + R_I P_C$$

where the subscripts indicate which coordinate system the item is represented with respect to, and R_I is the 3x3 matrix whose columns are X_I, Y_I and Z_I.

The process for displacements is rather different[3], which is the reason why we drew such a careful distinction between points and displacements.

Suppose that $D = P - Q$

Then we expect that both

$$D_C = P_C - Q_C$$
$$\text{and } D_I = P_I - Q_I$$

$$\begin{aligned}
\text{Now } D_I &= P_I - Q_I \\
&= O_I + R_I P_C - [O_I + R_I Q_C] \\
&= R_I P_C - R_I Q_C \\
&= R_I [P_C - Q_C] \\
&= R_I D_C
\end{aligned}$$

Mapping In

The algorithms for mapping in are easily derived by just solving the mapping out equations for P_C and D_C.

$$\begin{aligned}
P_C &:= R_I^{-1}[P_I - O_I] \\
D_C &:= R_I^{-1} D_I
\end{aligned}$$

[3]

Hierarchy of Transformations

There is a sequence of transformations, each more general than its successors in this list.

[Topological Similarity]
Projective Similarity
Affine Similarity
Similarity
Congruence (SBT)
Translations or Rotations
Identity

We are mainly concerned with the solid body transformations, but many of the operations we apply turn out to be invariant under more general groups.

Magic

At this point the rabbit is pulled out of the hat by pointing out that, because all of our entities are represented in terms of points and displacements, we can transform them by just transforming those pieces of the representation correctly.

Even coordinate systems themselves can be mapped in and out.

Making Coordinate Systems

The simple way to provide for the creation of coordinate systems is to build a small collection of kernel coordinate system methods, which provide, for example, translation and rotation about the implicit axis vectors.

These can then be combined, by using the fact that coordinate systems can themselves be mapped, to build more complex functions.

One kernel facility, which has proven to be more often required in practice than rotations, is a function to place the origin at a given point and orient the axes so that the X-axis passes through a second given point and the positive half of the XZ-plane passes through a third. This is a very natural way of defining how a camera should be positioned to get a good view of an object, and the image can then be computed by mapping points of the object into this coordinate system and then either ignoring the X dimension (for orthographic views) or dividing Z and Y by it (for perspective views).

Bounded Geometry

The lines and planes described above are **Unbounded**.

For many purposes we require to describe pieces of them.

Line Segment
Triangular Facet

Interrogations on Bounded objects

Every bounded object implies the unbounded object in which it lies, and so we may apply all of the unbounded computations to bounded inputs, expecting unbounded results.

The questions of 'In what does A meet/join B ?' are still well posed when the objects are bounded.

point x point $\rightarrow$ segment
point x point x point $\rightarrow$ triangle
triangle x triangle $\rightarrow$ segment
triangle x triangle x triangle $\rightarrow$ point
point x segment $\rightarrow$ triangle
triangle x segment $\rightarrow$ point

Now the result may be 'nothing' in a clean way, not due to numerical singularity, though singularity can still cause multiple solutions. Consider two disjoint triangular facets lying in the same plane.

In such cases, it is best to use an algorithm which avoids using the unbounded result, as it can introduce numerical problems unnecessarily. i.e. go directly for the bounded geometry rather than working through the unbounded objects and then trimming at the last minute.