[**Shapes and Coordinates**](https://0fps.net/2013/01/11/shapes-and-coordinates/)

Posted on [January 11, 2013](https://0fps.net/2013/01/11/shapes-and-coordinates/)by [mikolalysenko](https://0fps.net/author/mikolalysenko/)

In a [previous post](https://0fps.wordpress.com/2012/08/29/what-is-a-solid/), I talked a bit about solid modeling and discussed at a fairly high level what a solid object is.  This time I’m going to talk in generalities about how one actually represent shapes in practice.  My goal here is to say what these things are, and to introduce a bit of physical language which I think can be helpful when talking about these structures and their representations.

The parallel that I want to draw is that we should think of shape representations in much the same way that we think about coordinates.  A choice of coordinate system does not change the underlying reality of the thing we are talking about, but it can reveal to us different aspects of the nature of whatever it is we are studying.  Being conscious of the biases inherent in any particular point of view allows one to be more flexible when trying to solve some problem.

Two Types of Coordinates

I believe that there are two general ways to represent shapes: [**Eulerian** and **Lagrangian**](http://en.wikipedia.org/wiki/Lagrangian_and_Eulerian_specification_of_the_flow_field)**.**I’ll explain what this all means shortly, but as a brief preview here is a small table which contrasts these two approaches:

|  |  |  |
| --- | --- | --- |
|  | **Eulerian** | **Lagrangian** |
| **Coordinate System** | Global/fixed | Local/moving |
| **Modeling Paradigm** | Implicit | Parametric |
| **Transforms** | Passive | Active |

Eulerian (Implicit)

The first of these methods, or the Eulerian representation, seeks to describe shapes in terms of the space they embedded in.  That is shapes are described by maps from an ambient space into a classifying set:

f : \mathrm{Ambient}\:\mathrm{space} \to \mathrm{Classifying}\:\mathrm{space}

More concretely, suppose we are working in n-dimensional Euclidean space \mathbb{R}^n.  As we have already seen, it is possible to [describe solids using sublevel sets of functions](https://0fps.wordpress.com/2012/07/10/smooth-voxel-terrain-part-1/).  At a high level, the way this works is that we have some map f : \mathbb{R}^n \to S where S is some set of labels, for example the real line \mathbb{R}.  Then we can define our shape as the preimage of some subset of these labels, S_0 \subset S:

X = f^{-1}(S_0)

Again, if S = \mathbb{R} and S_0 = (-\infty, 0], then we get the usual definition of a sublevel set:

X = f^{-1}([-\infty, 0)) = \{ x \in \mathbb{R}^n : f(x) < 0 \} = V_0(f)

But these are of course somewhat arbitrary choices.  We could for example replace \mathbb{R} with the set S = \{ 0,1 \} of truth values, and we might consider defining our shape as the preimage of 1:

X = f^{-1}(1)

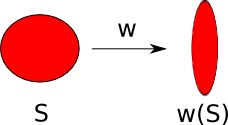
Which would be the set of all points in \mathbb{R}^n where *f* evaluates to 1.  In this situation *f* is what we could all point membership function.

Examples of Eulerian Representations

* [Implicit function modeling](http://hyperfun.org/wiki/doku.php?id=hyperfun:main)
* [Voxels](http://en.wikipedia.org/wiki/Voxel)
* [Constructive solid geometry](http://en.wikipedia.org/wiki/Constructive_solid_geometry)
* [Image bitmaps](http://en.wikipedia.org/wiki/Bitmap)

Motion in an Eulerian System

Say we have some Eulerian representation of our shape and we want to move it around.  One simple way to think about this is that we have the shape and we simply apply some map w : \mathbb{R}^n \to \mathbb{R}^n to warp it into a new position:

[](https://0fps.files.wordpress.com/2013/01/warp1.png)

Moving a shape by a map

I probably don’t need to convince you too much that motions are extremely useful in animation and physics, but they can also be a practical way to model shapes.  In fact, one simple way to construct more complicated shapes from simpler ones is to deform them along some kind of motion.  Now here is an important question:

**Question**: How do we apply a motion in an Eulerian coordinate system?

Or more formally, suppose that we have a set S \subseteq \mathbb{R}^n and we have some warp w : \mathbb{R}^n \to \mathbb{R}^n.  If S = f^{-1}( [-\infty, 0) is the 0-sublevel set of some potential function f, can we find another potential function h such that,

h^{-1}( [ -\infty, 0 ) ) = w(S)

If you’ve never thought about this before, I encourage you to pause for a moment to think about it carefully.

Done?  Ok, here is how we can arrive at the answer by pure algebraic manipulation.  Starting with the definition for S, we get:

h^{-1}( . ) = w( f^{-1}( . ) )

And taking [inverses](http://en.wikipedia.org/wiki/Inverse_function) on both sides,

h( x ) = f( w^{-1}( x ) )

Which is of course only valid when w is invertible.  So in summary,

*In an Eulerian coordinate system, to move a shape S = V_0(f) by w, we must precompose f with $w^{-1}$:*

w(S) = V_0(f \circ w^{-1})

Or said another way:

*Eulerian coordinate systems transform*[*passively*](http://en.wikipedia.org/wiki/Active_and_passive_transformation)*.*

Pros and Cons

There are many advantages to an Eulerian coordinate system.  For example, point wise operations like [Boolean set operations](http://en.wikipedia.org/wiki/Set_(mathematics)#Basic_operations) are particularly easy to compute.  (Just taking the min/max is sufficient to evaluate set intersection/union.)  On the other hand, because Eulerian coordinates are passive, operations like rendering or projection (which are secretly a kind of motion) can be much more expensive, since they require solving an inverse problem.  In general, this is as hard as root finding and so it is a non-trivial amount of work.  In 3D, there is also the bigger problem of storage, where representations like voxels can consume an enormous amount of memory.  However, in applications like image processing the advantages often outweigh the costs and so Eulerian representations are still widely used.

Lagrangian (Parametric)

Lagrangian coordinates are in many ways precisely the dual of Eulerian coordinates.  Instead of representing a shape as a map **from** an ambient space to some set of labels, they instead map a parametric space **into** the ambient space:

p : \mathrm{Parameter}\: \mathrm{space} \to \mathrm{Ambient}\: \mathrm{space}

The simplest example of a Lagrangian representation is a parametric curve, which typically acts by mapping an interval (like [0,1]) into some ambient space, say \mathbb{R}^n:

p(t) = ( x(t), y(t), z(t) )

But there are of course more sophisticated examples.  Probably the most common place where people bump into Lagrangian coordinates is in the representation of triangulated meshes.  A mesh is typically composed of two pieces of data:

1. A [cell complex](http://en.wikipedia.org/wiki/CW_complex) (typically specified by the vertex-face incidence data)
2. And an embedding fo the cell complex into \mathbb{R}^3

The embedding is commonly taken to be piecewise linear over each face.  There are of course many examples of Lagrangian representations:

Examples of Lagrangian Representations:

* [Meshes](http://en.wikipedia.org/wiki/Polygon_mesh)
* [Splines](http://en.wikipedia.org/wiki/Spline_(mathematics))
* [Point Clouds](http://en.wikipedia.org/wiki/Point_cloud)
* [Vector graphics](http://en.wikipedia.org/wiki/Vector_graphics)

Motion in Lagrangian Coordinates

Unlike in an Eulerian coordinate system, it is relatively easy to move an object in a Lagrangian system, since we can just apply a motion directly.  To see why, let p : P \to \mathbb{R}^n be a parameterization of a shape S and let w : \mathbb{R}^n \to \mathbb{R}^n be a motion; then obviously:

w(S) = w(p(P)) = (w \circ p)(P)

And so we conclude,

*Lagrangian coordinate systems transform actively.*

Pros and Cons

The main advantage to a Lagrangian formulation is that it is much more compact than the equivalent Eulerian form when the shape we are representing is lower dimensional than the ambient space.  This is almost always the case in graphics, where curves and surfaces are of course lower dimension than 3-space itself.  The other big advantage to Lagrangian coordinates is that they transform directly, which explains why they are much preferred in applications like 3D graphics.

On the other hand, Lagrangian coordinates are very bad at performing pointwise tests.  There is also a more subtle problem of validity.  While it is impossible to transform an Eulerian representation by a non-invertible transformation, this is quite easy to do in a Lagrangian coordinate system.  As a result, shapes can become non-manifold or self-intersecting which may break certain invariants and cause algorithms to fail.  This tends to make code that operates on Lagrangian systems much more complex than that which deals with Eulerian representations.