Revision: March 24, 2024

The Accelerator Lattice.jl Reference Manual

David Sagan

Contents

List of Figures

List of Tables

Part I

Lattice Construction and Manipulation

Chapter 1

Introduction, Overview, and Concepts

This chapter is an overview of, and an introduction to, the *AcceleratorLattice.jl* package which is part of the greater *Bmad-Julia* ecosystem of toolkits and programs for accelerator simulations. With *AcceleratorLattice.jl*, lattices, which can be used to describe such things as LINACs, storage rings, X-ray beam lines, can be constructed and manipulated. Tracking and lattice analysis (for example, calculating closed orbits and Twiss functions) is left to other packages in the *Bmad-Julia* ecosystem.

There are three main sources of documentation. One source is this manual which is written in LaTex and distributed in pdf format. A second source an introduction and overview guide written in Markdown. Finally, taking advantage of the

1.1 Acknowledgements

It is my pleasure to express appreciation to people who have contributed to this effort, and without whom, *Bmad-Julia* would only be a shadow of what it is today:

Étienne Forest (aka Patrice Nishikawa), Matthew Signorelli, Alexander Coxe, Oleksii Beznosov, Ryan Foussel, Auralee Edelen, Chris Mayes, Georg Hoffstaetter, Juan Pablo Gonzalez-Aguilera, Scott Berg, Dan Abell, Laurent Deniau, and Hugo Slepicka

1.2

The *Julia* language itself is used as the basis for constructing lattices with *AcceleratorLattice.jl*. Other simulation programs have similarly utilized the underlying programming language for constructing lattices[1, 4]. This is in marked contrast to many accelerator simulation programs such programs as MAD[3], Elegant[2], and the *Bmad* toolkit[5].

1.3 Using AcceleratorLattice.jl

AcceleratorLattice.jl is hosted on GitHub. The official repository is at

```
github.com/bmad-sim/AcceleratorLattice.jl
```

A `using` statement must be given before using *AcceleratorLattice.jl*

```
using AcceleratorLattice
```

1.4 Lattice Elements

The basic building block used to describe an accelerator is the lattice `element`. An element can be a physical thing that particles travel "through" like a bending magnet, a quadrupole or a Bragg crystal, or something like a `marker` element (§**??**) that is used to mark a particular point in the machine. Besides physical elements, there are `controller` elements that can be used for parameter control of other elements.

Chapter §3 lists the complete set of different element types that *Bmad* knows about.

In a lattice `branch` (§1.5), each element in the ordered array of elements are assigned an `element index` starting from one. The first element in a branch array is called `beginning_ele` (§3.1). This element is always included in every `branch` §1.5 and is used as a marker for the beginning of the `branch`. Additionally, every branch will have a final marker element (§**??**) named `end_ele`.

1.5 Lattice Branches

The next level up from an `element` is the `branch`. A `branch` contains an ordered sequence of lattice elements that a particle will travel through. A branch can represent a LINAC, X-Ray beam line, storage ring or anything else that can be represented as a simple ordered list of elements.

Chapter §**??** shows how a `branch` can be defined using `lines`.

Branches can be interconnected using `fork` elements (§**??**). This is used to simulate forking beam lines such as a connections to a transfer line, dump line, or an X-ray beam line. A `branch` from which other `branches` fork but is not forked to by any other `branch` is called a `root` branch. A branch that is forked to by some other branch is called a `downstream` branch.

There are two types of `branches`: `LordBranches` and `TrackingBranches`, Branches whose `Branch.type` are set to `LordBranch`

1.6 Lattice

A `lattice` (§1.6), has an array of `branches`. Each `branch` in this array has a name an is assigned an index starting from one. Branches are named after the line that defines the `branch`.

A `lattice` contains an array of `branches` that can be interconnected together to describe an entire machine complex. A `lattice` can include such things as transfer lines, dump lines, x-ray beam lines, colliding beam storage rings, etc. All of which can be connected together to form a coherent whole. In addition, a lattice may contain `controller elements` (Table 3.2) which can simulate such things as magnet power supplies and support structures like a girder supporting a section of magnets or an optical table supporting photonic elements.

Branches can be interconnected using `fork` and `photon_fork` elements (§**??**). This is used to simulate forking beam lines such as a connections to a transfer line, dump line, or an X-ray beam line. The `branch` from which other `branches` fork but is not forked to by any other `branch` is called a `root` branch.

A lattice may contain multiple `root branches`. For example, a pair of intersecting storage rings will generally have two `root` branches, one for each ring. The `use` statement (§??) in a lattice file will list the `root branches` of a lattice. To connect together lattice elements that are physically shared between branches, for example, the interaction region in colliding beam machines, `multipass` lines (§??) can be used.

1.7 Conventions

The following conventions

Evaluation is at upstream end: For lattice element parameters that are s-dependent, the evaluation location is the `upstream` edge of the element (§6.1.3). These parameters include the element's global position, the reference energy/momentum, and the s-position.

1.8 Convention Differences From Fortran Bmad

Certain conventions were changed

The old Fortran *Bmad* names for the coordinate systems (§6) was somewhat different and not always consistent. The `global` and `element body` names are the same but `machine` coordinates are called the `laboratory` in Fortran *Bmad*.

Evaluation was at the downstream end (§1.7) in Fortran *Bmad* not the upstream end.

Chapter 2

Constructing Lattices

2.1 Defining a Lattice Element

Chapter §?? gives a list of lattice elements defined by *AcceleratorLattice.jl*. Lattice elements are instantiated from structs which inherit from the abstract type `Lat`.

Elements are defined using the `@ele` macro. The general syntax is:

```
@ele eleName = eleType(param1 = val1, param2 = val2, ...)
```

where `eleName` is the name of the element, `eleType` is the type of element, `param1`, `param2`, etc. are parameter names and `val1`, `val2`, etc. are the parameter values. Example:

```
@ele qf = Quadrupole(L = 0.6, K1 = 0.370)
```

The `@ele` macro will construct a *Julia* variable with the name `eleName`. Additionally the element that this variable references will also hold `eleName` as the name of the element. So with this example, `qf.name` will be the string `"qf"`. If multiple elements are being defined, a single `@eles` macro can be used instead of multiple `@ele` macros. Example:

```
@eles begin
  s1 = Sextupole(L = ...)
  b2 = Bend(...)
  ...
end
```

2.2 Element Parameters

Generally, element parameters are grouped into "element parameter groups" structs which inherit from the abstract type `EleParameterGroup`. For example, the `LengthGroup` holds the length and s-positions of the element and is defined by:

```
@kwdef struct LengthGroup <: EleParameterGroup
  L::Number = 0
  s::Number = 0
  s_downstream::Number = 0
  orientation::Int = 1
end
```

The `kwdef` macro automatically defines a keyword-based constructor for `LengthGroup`. To see a list of all element parameter groups use the `subtypes(EleParamterGroup)` command. To see the components of a given group use the `fieldnames` function. For information on a given element parameter use the `info(::Symbol)` function. For example:

```
julia> info(:s_downstream)
  User name:       s_downstream
  Stored in:       LengthGroup.s_downstream
  Parameter type:  Number
  Units:           m
  Description:     Longitudinal s-position at the downstream end.
```

Notice that the argument to the `info` function is the symbol associated with the parameter. the "User name" is the name used when setting the parameter. This is discussed below. For most parameters, the User name and the name of the corresponding component in the element parameter group is the same. But there are exceptions. For example:

```
julia> info(:theta)
  User name:       theta_floor
  Stored in:       FloorPositionGroup.theta
  Parameter type:  Number
  Units:           rad
  Description:     Element floor theta angle orientation
```

2.3 Anatomy of an Element

The structs for all elements types contain exactly one component which is a Dict called `pdict` (short for "parameter dict") which is of type `Dict{Symbol,Any}`.

Element parameter groups §2.5 are stored in an element's `pdict`, with the key being the symbol associated with the group. For example, an element `ele` that has a `LengthGroup` will store this group at `ele.pdict[:LengthGroup]` and the length component `L` of this group is accessed by `ele.pdict[:LengthGroup].L`.

The `Base.setproperty` and `Base.getproperty` functions, which get called when the dot selection operator is used, have been overloaded for elements so that `ele.L` will get mapped to `ele.pdict[:LengthGroup].L`. Thus the following two statements both set the `s_downstream` parameter of an element named `q1`:

```
q1.pdict[:Length_group].s_downstream = q1.pdict[:Length_group].s +
                                            q1.pdict[:Length_group].L
q1.s_downstream = q1.s + q1.L
```

These two statements are not equivalent however. The difference is that when `s_downstream` is set using `q1.s_downstream`, the set is recorded by adding an entry to a Dict in the element at `ele.pdict[:changed]` where `ele` is the element whose parameter was changed. The key of the entry will be the symbol (`:s_downstream` in the present example) associated with the parameter and the value will be the old value of the parameter. When the `bookkeeper(::Lat)` function is called, the bookkeeping code will use the entries in `ele.pdict[:changed]` to limit the bookkeeping to what is necessary and thus minimize computing time. Knowing what has been changed is also important in resolving what parameters need to be changed. For example, if the bend `angle` of a bend is changed, the bookkeeping code will set the bending strength `g` using the equation `g` = `angle` / `L`. If, instead, if `g` is changed, the bookkeeping code will set `angle` appropriately.

2.4 Bookkeeping and Dependent Element Parameters

After lattice parameters are changed, the function `bookkeeper(::Lat)` needs to be called so that dependent parameters can be updated. Since bookkeeping can take a significant amount of time if bookkeeping is done every time a change to the lattice is made, and since there is no good way to tell when bookkeeping should be done, After lattice expansion, `bookkeeper(::Lat)` is never called directly by *AcceleratorLattice.jl* functions and needs to be called by the User when appropriate (generally before tracking or other computations are done).

Broadly there are two types of dependent parameters: intra-element dependent parameters where the changed parameters and the dependent parameters are all within the same element and cascading dependent parameters where changes to one element cause changes to parameters of elements downstream.

The cascading dependencies are:

s-position dependency: Changes to an elements length `L` or changes to the beginning element's `s` parameter will result in the s-positions of all downstream elements changing.

Reference energy dependency: Changes to the be beginning element's reference energy (or equivilantly the referece momentum), or changes to the `voltage` of an `LCavity` element will result in the reference energy of all downstream elements changing.

Global position dependency: The position of a lattice element in the global coordinate system (§6.2) is affected by a) the lengths of all upstream elements, b) the bend angles of all upstream elements, and c) the position in global coordinates of the beginning element.

2.5 Defining a Lattice Element Type

All lattice element types like `Quadrupole`, `Marker`, etc. are subtypes of the abstract type `Ele`. To construct a new type, use the `construct_ele_type` macro. Example:

```
  @construct_ele_type MyEleType
```

Chapter 3

Lattice Element Types

This chapter discusses the various types of elements available in *Bmad-Julia*. Table 3.1 lists the elements that can be tracked through. Table 3.2 lists the `controller` element types that can be used for parameter control of other elements.

Element	*Section*	*Element*	*Section*
`ACKicker`	??	`Marker`	??
`BeamBeam`	??	`Mask`	??
`BeginningEle`	3.1	`Match`	??
`Bend`	??	`Multipole`	??
`Collimator`	??	`NullEle`	??
`Converter`	??	`Octupole`	??
`CrabCavity`	??	`Patch`	??
`Custom`	??	`Pipe`	??
`Drift`	??	`Quadrupole`	??
`EGun`	??	`RFbend`	??
`ElSeparator`	??	`RFcavity`	??
`EMField`	??	`SadMult`	??
`Fiducial`	??	`Sextupole`	??
`FloorShift`	??	`Solenoid`	??
`Foil`	??	`Taylor`	??
`Fork`	??	`ThickMultipole`	??
`Instrument`	??	`Undulator`	??
`Kicker`	??	`UnionEle`	??
`Lcavity`	??	`Wiggler`	??

Table 3.1: Table of element types suitable for use with charged particles. Also see Table 3.2

Element	*Section*	*Element*	*Section*
`Controller`	??	`Ramper`	??
`Girder`	??		

Table 3.2: Table of controller elements.

3.1 Beginning_Ele

A `beginning_ele` element, named “`BEGINNING`”, is placed at the beginning of every branch (§1.5) of a lattice to mark the start of the branch. The `beginning_ele` always has element index 0 (§**??**). The creation of this `beginning_ele` element is automatic and it is not permitted to define a lattice with `beginning_ele` elements at any other position.

The attributes of the `beginning_ele` element in the root branch are are generally set using `beginning` (§**??**) statements or line parameter (§**??**) statements. [The attributes of other `beginning_ele` elements are set solely with line parameter statements.]

If the first element after the `beginning_ele` element at the start of a branch is reversed (§**??**), the `beginning_ele` element will be marked as reversed so that a reflection patch is not needed in this circumstance.

See §**??** for a full list of element attributes.

Chapter 4

Customizing Lattices

`Custom Lattice Element Parameters`

Custom parameters may be added to lattice elements but methods need to be created to tell *AcceleratorLattice.jl* how to handle these parameters.

* Define element parameter group

* Need to extend: ele_param_info_dict param_groups_list param_group_info

`Custom Lattice Elements`

* Need to extend: ele_param_groups

Chapter 5

Design Decisions

This chapter discusses some of the design decisions that were made in the planning of *Accelerator-Lattice.jl*. Hopefully this information will be useful as *AcceleratorLattice.jl* is developed in the future. The design of *AcceleratorLattice.jl* is heavily influenced by the decades of experience constructing and maintaining *Bmad*— both in terms of what works and what has not worked.

First a clarification. The name *Bmad* can be used in two senses. There is *Bmad* the software toolkit that can be used to create simulation programs. But *Bmad* can also be used to refer to the ecosystem of toolkit and *Bmad* based programs that have been developed over the years — the most heavily used program being Tao. In the discussion below, *Bmad* generally refers to the toolkit since it is the toolkit that defines the syntax for *Bmad* lattice files.

Bmad history: To understand *AcceleratorLattice.jl* it helps to understand some of the history of *Bmad*. The *Bmad* toolkit started out as a modest project for calculating Twiss parameters and closed orbits within online control programs for the Cornell CESR storage ring. As such, the lattice structure was simply an array of elements. That is, multiple branches could not be instantiated. And tracking was very simple — there was only one tracking method, symplecticity was ignored and ultra-relativistic and paraxial approximations were used. *Bmad* has come a long way from the early days but design decisions made early on still haunt the *Bmad* toolkit.

Separation of tracking and lattice description: One of the first *AcceleratorLattice.jl* design decisions was to separate, as much as possible, particle tracking and lattice description. This decision was inspired by the PTC code of Etienne Forest. The fact that *Bmad* did not make this separation complicated *Bmad*'s lattice element structure, the `ele_struct`, to the extent that the `ele_struct` is the most complicated structure in all of *Bmad*. And having complicated structures is an impediment to code sustainability. The lack of a separation in *Bmad* also made bookkeeping more complicated in cases where, for example, Twiss parameters were to be calculated under differing conditions (EG varing initial particle positions) but the `ele_struct` can only hold Twiss parameters for one specific condition.

Lattice branches: The organization of the lattice into branches with each branch having an array of elements has worked very well with *Bmad* and so is used with *AcceleratorLattice.jl*. The relatively minor difference is that with *AcceleratorLattice.jl* the organization of the branches is more logical with multiple lord branches with each lord branch containing only one type of lord.

Type stability: Type stability is *not* a major concern with *AcceleratorLattice.jl*. The reason being that compared to the time needed for tracking and track analysis, lattice instantiation and manipulation does not take an appreciable amount of time. For tracking, where computation time is a hugh consideration, an interface layer can be used to translate lattice parameters to a type stable form. Of much greater importance is flexibility of *AcceleratorLattice.jl* to accomodate changing needs and software sustainability.

Lattice element structure: All lattice element structs are very simple: They contain a single Dict and all element information is stored within this Dict. This means that there is no restriction as to what can be stored in an element adding custom information to an element simple. And the ability to do customization easily is very important.

Within an element Dict, for the most part, parameters are grouped into "element group" structs. A flattened structure, that is, without the element group structs, would be the correct strategy if the number of possible parameters for a given element type was not as large as it is. However, the parameterization of an element can be complicated. For example, a field table describing the field in an element has a grid of field points plus parameters to specify the distance between points, the frequency if the field is oscillating, etc. In such a case, where the number of parameters is large, and with the parameters falling into logical groups, using substructures if preferred.

Part II

Conventions and Physics

Chapter 6

Coordinates

Bmad uses three coordinate systems as illustrated in Fig. 6.1. First, the `global` (also called "`floor`") coordinates are independent of the accelerator. Thus such things as the building the accelerator is in may be described using `global` coordinates.

It is not convenient to describe the position of the beam using the `global` coordinate system so a "`machine`" coordinate system is used (§6.1). This curvilinear coordinate system defines the nominal position of the lattice elements. The relationship between the `machine` and `global` coordinate systems is described in §6.2.

The "nominal" position of a lattice element is the position of the element without what are called "misalignments" (that is, position and orientation shifts). Each lattice element has "`element body`" (or just "`body`") coordinates which are attached to the physical element. That is, the electric and magnetic fields of an element are described with respect to `body` coordinates. If there are no misalignments, the `body` coordinates are aligned with `local` coordinates which are the same as `machine` coordinates except that $s = 0$ in `local` coordinates is at the beginning of the non-misaligned element and $s = 0$ in machine coordinates is at the beginning of the line. The transformation between `machine` and `body` coordinates is given in §6.3. The $x = y = 0$ curved line of the machine coordinate system is known as the "`reference orbit`".

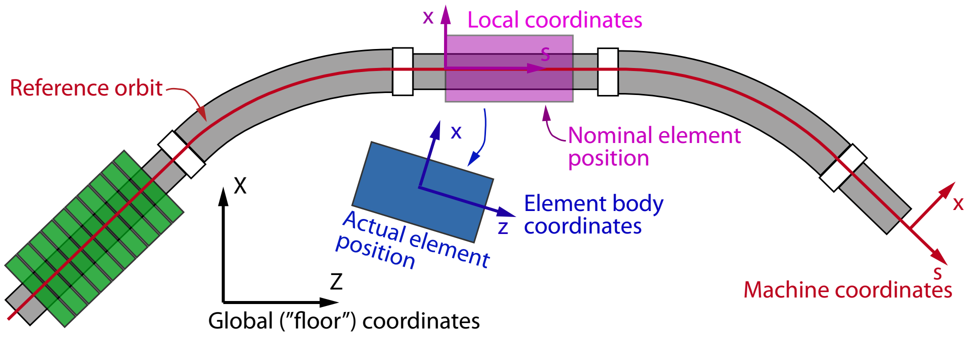

Figure 6.1: The `global` (or "`floor`") coordinate system is independent of the accelerator. The `machine` curvilinear coordinate system follows the bends of the accelerator. Each lattice element has `element body` coordinates which, if the element is not "misaligned" is the same as the `local` coordinates. `Local` coordinates are the same as `machine` coordinates except that $s = 0$ in `local` coordinates is at the beginning of the non-misaligned element and $s = 0$ in machine coordinates is at the beginning of the line. The $x = y = 0$ curved line of the machine coordinate system is known as the "reference orbit".

6.1 Machine Coordinates and Reference Orbit

6.1.1 The Reference Orbit

The `local reference orbit` is the curved path used to define a coordinate system for describing a particle's position as shown in Fig. 6.2. The reference orbit is also used for orientating lattice elements in space. At a given time t, a particle's position can be described by a point $\mathcal{O}$ on the reference orbit a distance s relative to the reference orbit's zero position plus a transverse (x, y) offset. The point $\mathcal{O}$ on the reference orbit is used as the origin of the local (x, y, z) coordinate system with the z–axis tangent to the reference orbit. The z–axis will generally be pointing in the direction of increasing s (Fig. 6.2A) but, as discussed below, will point counter to s for elements that are `reversed` (Fig. 6.2B). The x and y–axes are perpendicular to the reference orbit and, by construction, the particle is always at $z = 0$. The coordinate system so constructed is called the “`local coordinate system`” or sometimes the “`machine coordinate system`” when there is need to distinguish it from the “`element body coordinate system`” (§6) which is attached to the physical element. There is a separate reference orbit for each branch (§1.5) of a lattice.

Notice that, in a `wiggler`, the reference orbit, which is a straight line, does *not* correspond to the orbit that any actual particle could travel. Typically the physical element is centered with respect to the reference curve. However, by specifying offsets, pitches or a tilt (See §**??**), the physical element may be arbitrarily shifted with respect to its reference curve. Shifting a physical magnet with respect to its reference curve generally means that the reference curve does *not* correspond to the orbit that any actual particle could travel.

Do not confuse this reference orbit (which defines the local coordinate system) with the reference orbit about which the transfer maps are calculated (§**??**). The former is fixed by the lattice while the latter can be any arbitrary orbit.

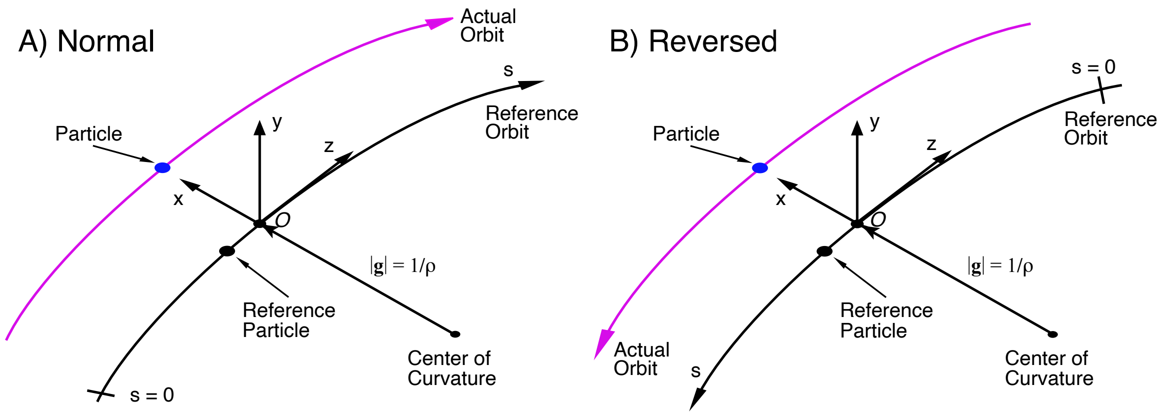

Figure 6.2: The local reference coordinate system. By construction, a particle's z coordinate is zero. This is not to be confused with the phase space z coordinate (§6.4.2). The curvature vector $\mathbf{g}$ lies in the x-y plane and has a magnitude of $1/\rho$ where ρ is the bending radius. A) The z-axis will normally be parallel to the s-axis. B) For `reversed` elements it will be antiparallel. In both cases, the particle and reference particle are traveling in the direction of greater s.

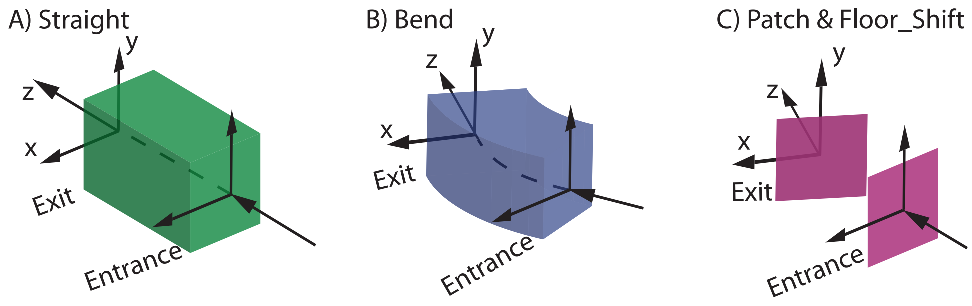

Figure 6.3: Lattice elements can be imagined as "LEGO blocks" which fit together to form the reference orbit along with the machine coordinate system. How elements join together is determined in part by their entrance and exit coordinate frames. A) For straight line elements the entrance and exit frames are colinear. B) For bends elements, the two frames are rotated with respect to each other. C) For `patch` and `floor_shift` elements the exit frame may be arbitrarily positioned with respect to the entrance frame.

6.1.2 Element Entrance and Exit Coordinates

One way of thinking about the reference orbit and the machine coordinates is to imagine that each element is like a LEGO block with an "`entrance`" and an "`exit`" coordinate frame as illustrated in Fig. 6.3[1]. These coordinate frames are attached to the element. that is, things like electric and magnetic fields, apertures, etc., are described with respect to the entrance and exit coordinates. Thus, for example, the `e1` edge of a bend (§??) is always at the `entrance` face and the `e2` is always at the `exit` face. Most elements have a "straight" geometry as shown in Fig. 6.3A. That is, the reference orbit through the element is a straight line segment with the x and y axes always pointing in the same direction. For a `bend` element (§??), the reference orbit is a segment of a circular arc as shown in Fig. 6.3B. With the `ref_tilt` parameter of a bend set to zero, the rotation axis between the entrance and exit frames is parallel to the y-axis (§6.2). For `patch` (§??), and `floor_shift` (§??) elements, the exit face can can arbitrarily oriented with respect to the entrance end. In this case, the reference orbit between the entrance and exit faces is not defined.

6.1.3 Reference Orbit and Machine Coordinates Construction

Assuming for the moment that there are no `fiducial` elements present, the construction of the reference orbit starts at the `beginning_ele` element (§3.1) at the start of a branch. If the branch is a `root` branch (§1.6), The orientation of the beginning element within the global coordinate system (§6) can be set via the appropriate positioning statements (§??). If the branch is not a `root` branch, the position of the beginning element is determined by the position of the `fork` or `photon_fork` element from which the branch forks from. Unless set otherwise in the lattice file, $s = 0$ at the `beginning_ele` element.

If there are `fiducial` elements, the machine coordinates are constructed beginning at these elements.

Once the beginning element in a branch is positioned, succeeding elements are concatenated together to form the machine coordinates. All elements have an "`upstream`" and a "`downstream`" end as shown in Fig. 6.4A. The `downstream` end of an element is always farther (at greater s) from the beginning element than the `upstream` end of the element. Particles travel in the $+s$ direction, so particles will enter an element at the upstream end and exit at the downstream end.

[1]Thanks to Dan Abell for this analogy.

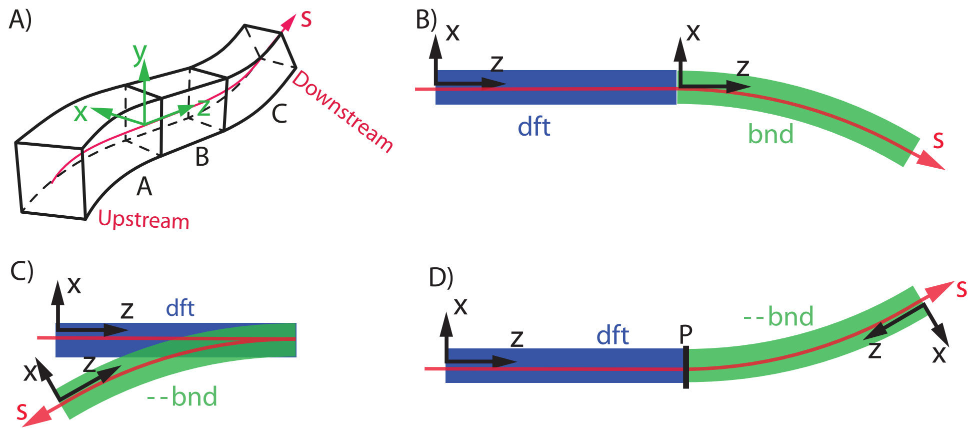

Figure 6.4: A) The machine coordinates are constructed by connecting the `downstream` reference frame of one element with the `upstream` reference frame of the next element in the branch. Coordinates shown is for the mating of element `A` to element B. B) Example with drift element `dft` followed by a bend `bnd`. Both elements are unreversed. C) Similar to (B) but in this case element `bnd` is reversed. D) Similar to (C) but in this case a reflection patch has been added in between `dft` and `bnd`. In (B), (C), and (D) the (x, z) coordinates are drawn at the `entrance` end of the elements. The y coordinate is always out of the page.

Normally, the `upstream` end is the same as the element's `entrance` end (Fig. 6.3) and the `downstream` end is the same as the element's `exit` end. However, if an element is reversed (§??), the element's `exit` end will be `upstream` end and the element's `entrance` end will be the `downstream` end. That is, for a reversed element, particles traveling downstream will enter at the element's `exit` end and will exit at the `entrance` end.

The procedure to connect elements together to form the machine coordinates is to mate the downstream reference frame of the element with the upstream reference frame of the next element in the branch so that, without misalignments, the (x, y, z) axes coincide[2]. This is illustrated in Fig. 6.4. Fig. 6.4A shows the general situation with the downstream frame of element `A` mated to the upstream frame of element `B`. Figures 6.4B-C show branches constructed from the following lattice file:

```
@ele dft = drift(L = 2)
@ele bnd = bend(l = 2, g = pi/12)
@ele p = patch(x_pitch = pi)                ! Reflection patch.
BL = beamline("BL", [dft, bnd])             ! No reversal.
CL = beamline("CL", [dft, reverse(bnd)])    ! Illegal. Do not use!
DL = beamline("DL", [dft, p, reverse(bnd)]) ! Valid.
```

The (x, z) coordinates are drawn at the entrance end of the elements and z will always point towards the element's exit end. Fig. 6.4B shows the branch constructed from `BL` containing an unreversed drift named `dft` connected to an unreversed bend named `bnd`. Fig. 6.4C shows the branch constructed from `CL`. This is like `BL` except here element `bnd` is reversed. This gives an unphysical situation since a particle traveling through `dft` will "fall off" when it gets to the end. Fig. 6.4D shows the branch constructed from `DL`. Here a "`reflection`" patch `P` (§6.2.6) has been added to get a plausible geometry. The patch rotates the coordinate system around the y-axis by 180°(leaving the y-axis invariant). It is always the case that a reflection patch is needed between reversed and unreversed elements

Notes:

[2]If there are misalignments, the `entrance` and `exit` frames will move with the element. However, the `upstream` and `downstream` frames, along with the reference orbit and machine coordinates, will not move.

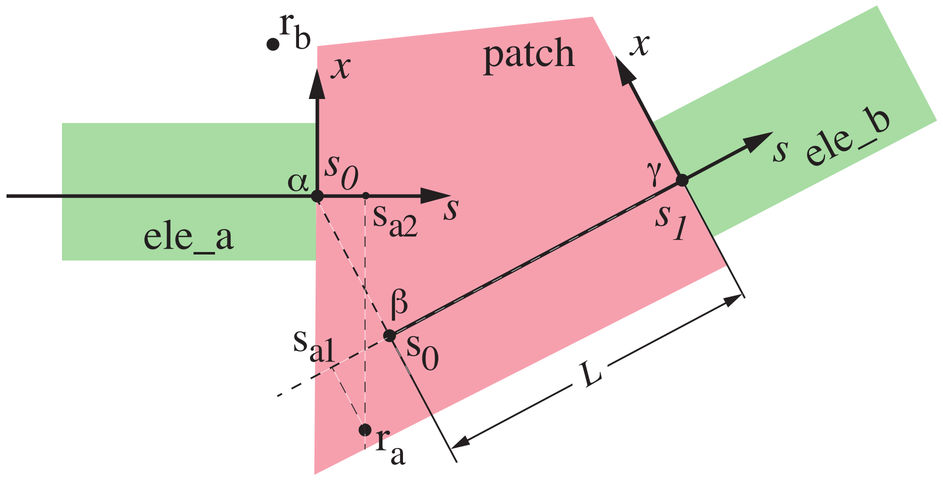

Figure 6.5: The local reference coordinates in a `patch` element. The `patch` element, shown schematically as an irregular quadrilateral, is sandwiched between elements `ele_a` and `ele_b`. `L` is the length of the `patch`. In this example, the `patch` has a finite `x_pitch`.

- If the first element after the `beginning_ele` element at the start of a branch is reversed, the `beginning_ele` element will be marked as reversed so that a reflection patch is not needed in this circumstance.
- Irrespective of whether elements are reversed or not, the machine (x, y, z) coordinate system at all s-positions will always be a right-handed coordinate system.
- Care must be take when using reversed elements. For example, if the field of the `bnd` element in `B_line` is appropriate for, say, electrons, that is, electrons will be bent in a clockwise fashion going through `bnd`, then an electron going through `D_line` will be lost in the bend (the y-axis and hence the field is in the same direction for both cases so electrons will still be bent in a clockwise fashion but with `D_line` a particle needs to be bent counterclockwise to get through the bend). To get a particle through the bend, positrons must be used.
- A reflection patch that rotated the coordinates, for example, around the x-axis by 180°(by setting `y_pitch` to `pi`) would also produce a plausible geometry.

6.1.4 Patch Element Local Coordinates

Generally, if a particle is reasonably near the reference orbit, there is a one-to-one mapping between the particle's position and (x, y, s) coordinates. A `patch` (§**??**) elements with a non-zero `x_pitch` or non-zero `y_pitch` breaks the one-to-one mapping. This is illustrated in Fig. 6.5. The `patch` element, shown schematically as an, irregular quadrilateral, is sandwiched between elements `ele_a` and `ele_b`. The local coordinate system with origin at α are the coordinates at the end of `ele_a`. The coordinates at the end of the `patch` has its origin labeled γ. By convention, the length of the patch `L` is taken to be the longitudinal distance from α to γ with the `patch`'s exit coordinates defining the longitudinal direction. The "beginning" point of the `patch` on the reference orbit a distance `L` from point γ is labeled β in the figure.

In the local (x, y, s) coordinate system a particle at α will have some value $s = s_0$. A particle at point β will have the same value $s = s_0$ and a particle at γ will have $s = s_1 = s_0 + L$. A particle at point r_a

in Fig. 6.5 illustrates the problem of assigning (x, y, s) coordinates to a given position. If the particle is considered to be within the region of `ele_a`, the particle's s position will be s_{a2} which is greater than the value s_0 at the exit end of the element. This contradicts the expectation that particles within `ele_a` will have $s \le s_0$. If, on the other hand, the particle is considered to be within the `patch` region, the particle's s position will be s_{a1} which is less than the value s_0 at the entrance to the patch. This contradicts the expectation that a particles within the `patch` will have $s \ge s_0$.

To resolve this problem, *Bmad* considers a particle at position r_a to be within the `patch` region. This means that there is, in theory, no lower limit to the s-position that a particle in the `patch` region can have. This also implies that there is a discontinuity in the s-position of a particle crossing the exit face of `ele1`. Typically, when particles are translated from the exit face of one element to the exit face of the next, this `patch` problem does not appear. It only appears when the track between faces is considered.

Notice that a particle at position r_b in Fig. 6.5 can simultaneously be considered to be in either `ele_a` or the `patch`. While this creates an ambiguity it does not complicate tracking.

6.2 Global Coordinates

The Cartesian `global` coordinate system, also called the 'floor" coordinate system, is the coordinate system "attached to the earth" that is used to describe the local coordinate system. Following the *MAD* convention, the `global` coordinate axis are labeled (X, Y, Z). Conventionally, Y is the "vertical" coordinate and (X, Z) are the "horizontal" coordinates. To describe how the local coordinate system is oriented within the global coordinate system, each point on the s-axis of the local coordinate system

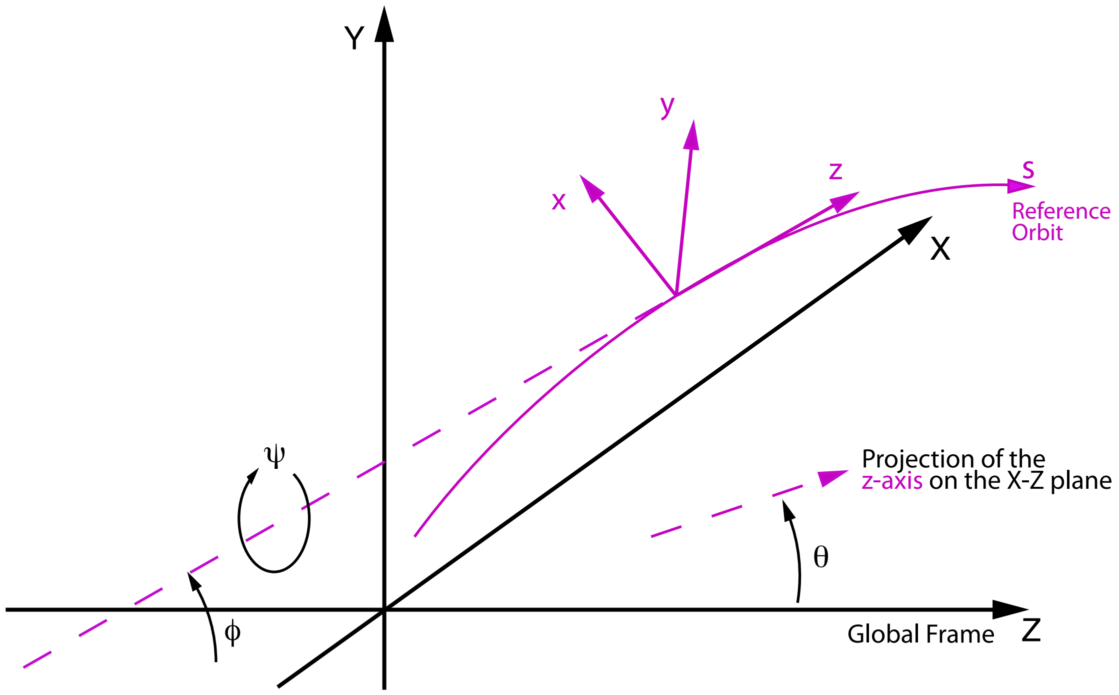

Figure 6.6: The local (reference) coordinate system (purple), which is a function of s along the reference orbit, is described in the global coordinate system (black) by a position $(X(s), Y(s), Z(s))$ and and by angles $\theta(s)$, $\phi(s)$, and $\psi(s)$.

is characterized by its (X, Y, Z) position and by three angles $\theta(s)$, $\phi(s)$, and $\psi(s)$ that describe the orientation of the local coordinate axes as shown in Fig. 6.6. These three angles are defined as follows:

$\theta(s)$ **Azimuth (yaw) angle:** Angle in the (X, Z) plane between the Z–axis and the projection of the z–axis onto the (X, Z) plane. Corresponds to the `x_pitch` element attribute (§**??**). A positive angle of $\theta = \pi/2$ corresponds to the projected z–axis pointing in the positive X direction.

$\phi(s)$ **Pitch (elevation) angle:** Angle between the z–axis and the (X, Z) plane. Corresponds to the `y_pitch` element attribute (§**??**). A positive angle of $\phi = \pi/2$ corresponds to the z–axis pointing in the positive Y direction.

$\psi(s)$ **Roll angle:** Angle of the x–axis with respect to the line formed by the intersection of the (X, Z) plane with the (x, y) plane. Corresponds to the `tilt` element attribute (§**??**). A positive ψ forms a right–handed screw with the z–axis.

By default, at $s = 0$, the reference orbit's origin coincides with the (X, Y, Z) origin and the x, y, and z axes correspond to the X, Y, and Z axes respectively. If the lattice has no vertical bends (the `ref_tilt` parameter (§**??**) of all bends are zero), the y–axis will always be in the vertical Y direction and the x–axis will lie in the horizontal (X, Z) plane. In this case, θ decreases as one follows the reference orbit when going through a horizontal bend with a positive bending angle. This corresponds to x pointing radially outward. Without any vertical bends, the Y and y axes will coincide, and ϕ and ψ will both be zero. The `beginning` statement (§**??**) in a lattice file can be use to override these defaults.

Following *MAD*, the global position of an element is characterized by a vector $\mathbf{V}$

$$\mathbf{V} = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{6.1}$$

The orientation of an element is described by a unitary rotation matrix $\mathbf{W}$. The column vectors of $\mathbf{W}$ are the unit vectors spanning the local coordinate axes in the order (x, y, z). $\mathbf{W}$ can be expressed in terms of the orientation angles θ, ϕ, and ψ via the formula

$$\begin{aligned} \mathbf{W} &= \mathbf{R}_y(\theta)\, \mathbf{R}_{-x}(\phi)\, \mathbf{R}_z(\psi) \qquad (6.2) \\ &= \begin{pmatrix} \cos\theta\cos\psi - \sin\theta\sin\phi\sin\psi & -\cos\theta\sin\psi - \sin\theta\sin\phi\cos\psi & \sin\theta\cos\phi \\ \cos\phi\sin\psi & \cos\phi\cos\psi & \sin\phi \\ -\cos\theta\sin\phi\sin\psi - \sin\theta\cos\psi & \sin\theta\sin\psi - \cos\theta\sin\phi\cos\psi & \cos\theta\cos\phi \end{pmatrix} \end{aligned}$$

where

$$\mathbf{R}_y(\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}, \quad \mathbf{R}_{-x}(\phi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{pmatrix}, \quad \mathbf{R}_z(\psi) = \begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{6.3}$$

Notice that $\mathbf{R}_{-x}(\phi)$, for positive ϕ, represents a rotation around the negative x-axis. Also notice that these are Tait-Bryan angles and not Euler angles.

An alternative representation of the $\mathbf{W}$ matrix (or any other rotation matrix) is to specify the axis $\mathbf{u}$ (normalized to 1) and angle of rotation β

$$\mathbf{W} = \begin{pmatrix} \cos\beta + u_x^2\,(1-\cos\beta) & u_x\,u_y\,(1-\cos\beta) - u_z\sin\beta & u_x\,u_z\,(1-\cos\beta) + u_y\sin\beta \\ u_y\,u_x\,(1-\cos\beta) + u_z\sin\beta & \cos\beta + u_y^2\,(1-\cos\beta) & u_y\,u_z\,(1-\cos\beta) - u_x\sin\beta \\ u_z\,u_x\,(1-\cos\beta) - u_y\sin\beta & u_z\,u_y\,(1-\cos\beta) + u_x\sin\beta & \cos\beta + u_z^2\,(1-\cos\beta) \end{pmatrix} \tag{6.4}$$

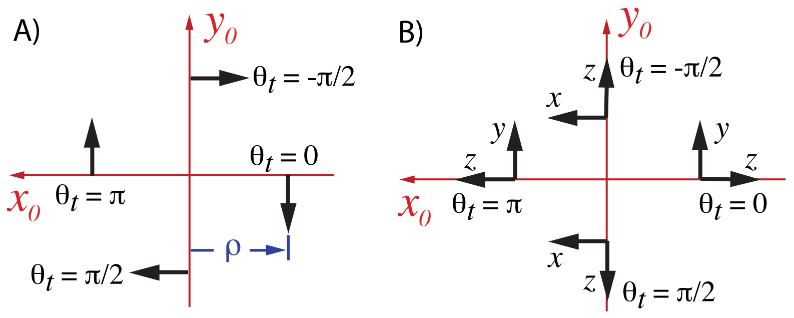

Figure 6.7: A) Rotation axes (bold arrows) for four different `ref_tilt` angles of $\theta_t = 0$, $\pm\pi/2$, and π. (x_0, y_0, z_0) are the local coordinates at the entrance end of the bend with the z_0 axis being directed into the page. Any rotation axis will be displaced by a distance of the bend radius `rho` from the origin. B) The (x, y, z) coordinates at the exit end of the bend for the same four `ref_tilt` angles. In this case the bend angle is taken to be $\pi/2$.

6.2.1 Lattice Element Positioning

Bmad, again following *MAD*, computes **V** and **W** by starting at the first element of the lattice and iteratively using the equations

$$\mathbf{V}_i = \mathbf{W}_{i-1}\,\mathbf{L}_i + \mathbf{V}_{i-1}, \tag{6.5}$$

$$\mathbf{W}_i = \mathbf{W}_{i-1}\,\mathbf{S}_i \tag{6.6}$$

$\mathbf{L}_i$ is the displacement vector for the i^{th} element and matrix $\mathbf{S}_i$ is the rotation of the local reference system of the exit end with respect to the entrance end. For clarity, the subscript i in the equations below will be dripped. For all elements whose reference orbit through them is a straight line, the corresponding **L** and **S** are

$$\mathbf{L} = \begin{pmatrix} 0 \\ 0 \\ L \end{pmatrix}, \quad \mathbf{S} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{6.7}$$

Where L is the length of the element.

For a `bend`, the axis of rotation is dependent upon the bend's `ref_tilt` angle (§??) as shown in Fig. 6.7A. The axis of rotation points in the negative y_0 direction for `ref_tilt` $= 0$ and is offset by the bend radius `rho`. Here (x_0, y_0, z_0) are the local coordinates at the entrance end of the bend with the z_0 axis being directed into the page in the figure. For a non-zero `ref_tilt`, the rotation axis is itself rotated about the z_0 axis by the value of `ref_tilt`. Fig. 6.7B shows the exit coordinates for four different values of `ref_tilt` and for a bend angle `angle` of $\pi/2$. Notice that for a bend in the horizontal $X - Z$ plane, a positive bend `angle` will result in a decreasing azimuth angle θ.

For a bend, **S** is given using Eq. (6.4) with

$$\mathbf{u} = (-\sin\theta_t, -\cos\theta_t, 0)$$
$$\beta = \alpha_b \tag{6.8}$$

where θ_t is the `ref_tilt` angle. The **L** vector for a `bend` is given by

$$\mathbf{L} = \mathbf{R}_z(\theta_t)\,\widetilde{\mathbf{L}}, \quad \widetilde{\mathbf{L}} = \begin{pmatrix} \rho(\cos\alpha_b - 1) \\ 0 \\ \rho\,\sin\alpha_b \end{pmatrix} \tag{6.9}$$

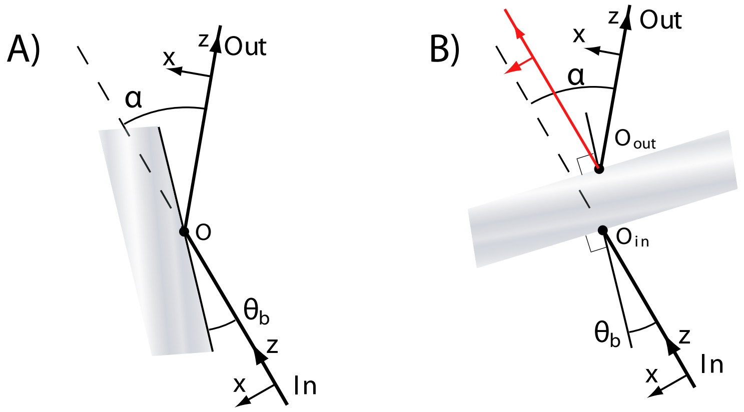

Figure 6.8: Mirror and crystal geometry. The geometry shown here is appropriate for a `ref_tilt` angle of $\theta_t = 0$. θ_g is the bend angle of the incoming (entrance) ray, and α_b is the total bend angle of the reference trajectory. A) Geometry for a mirror or a Bragg crystal. Point $\mathcal{O}$ is the origin of both the local coordinates just before and just after the reflection/diffraction. B) Geometry for a Laue crystal. Point $\mathcal{O}_{out}$ is the origin of the coordinates just after diffraction is displaced from the origin $\mathcal{O}_{in}$ just before diffraction due to the finite thickness of the crystal. here the bend angles are measured with respect to the line that is in the plane of the entrance and exit coordinates and perpendicular to the surface. For Laue diffraction, the user has the option of using the undiffracted beam (shown in red) as the reference trajectory.

where α_b is the bend `angle` (§**??**) and ρ being the bend radius (`rho`). Notice that since $\mathbf{u}$ is perpendicular to z, the curvilinear reference coordinate system has no "torsion". That is, it is a Frenet-Serret coordinate system.

Note: An alternative equation for $\mathbf{S}$ for a bend is

$$\mathbf{S} = \mathbf{R}_z(\theta_t)\, \mathbf{R}_y(-\alpha_b)\, \mathbf{R}_z(-\theta_t) \tag{6.10}$$

The bend transformation above is so constructed that the transformation is equivalent to rotating the local coordinate system around an axis that is perpendicular to the plane of the bend. This rotation axis is invariant under the bend transformation. For example, for $\theta_t = 0$ (or π) the y-axis is the rotation axis and the y-axis of the local coordinates before the bend will be parallel to the y-axis of the local coordinates after the bend as shown in Fig. 6.7. That is, a lattice with only bends with $\theta_t = 0$ or π will lie in the horizontal plane (this assuming that the y-axis starts out pointing along the Y-axis as it does by default). For $\theta_t = \pm\pi/2$, the bend axis is the x-axis. A value of $\theta_t = +\pi/2$ represents a downward pointing bend.

6.2.2 Position Transformation When Transforming Coordinates

A point $\mathbf{Q}_g = (X, Y, Z)$ defined in the global coordinate system, when expressed in the coordinate system defined by $(\mathbf{V}, \mathbf{W})$ is

$$\mathbf{Q}_{VW} = \mathbf{W}^{-1}\, (\mathbf{Q}_g - \mathbf{V}) \tag{6.11}$$

This is essentially the inverse of Eq. (6.5). That is, vectors propagate inversely to the propagation of the coordinate system.

Using Eq. (6.11) with Eqs. (6.5), and (6.6), the transformation of a particle's position $\mathbf{q} = (x, y, z)$ and momentum $\mathbf{P} = (P_x, P_y, P_z)$ when the coordinate frame is transformed from frame $(\mathbf{V}_{i-1}, \mathbf{W}_{i-1})$ to frame $(\mathbf{V}_i, \mathbf{W}_i)$ is

$$\mathbf{q}_i = \mathbf{S}_i^{-1} \left(\mathbf{q}_{i-1} - \mathbf{L}_i \right), \tag{6.12}$$

$$\mathbf{P}_i = \mathbf{S}_i^{-1} \, \mathbf{P}_{i-1} \tag{6.13}$$

Notice that since $\mathbf{S}$ (and $\mathbf{W}$) is the product of orthogonal rotation matrices, $\mathbf{S}$ is itself orthogonal and its inverse is just the transpose

$$\mathbf{S}^{-1} = \mathbf{S}^T \tag{6.14}$$

6.2.3 Crystal and Mirror Element Coordinate Transformation

A `crystal` element (§**??**) diffracts photons and a `mirror` element (§**??**) reflects them. For a crystal setup for Bragg diffraction, and for a mirror, the reference orbit is modeled as a zero length bend with $\widetilde{\mathbf{L}} = (0, 0, 0)$, as shown in Fig. 6.8A. Shown in the figure is the geometry appropriate for a `ref_tilt` angle of $\theta_t = 0$ (the rotation axis is here the y-axis). Since the mirror or crystal element is modeled to be of zero length, the origin points (marked $\mathcal{O}$ in the figure) of the entrance and exit local coordinates are the same. For Laue diffraction, the only difference is that $\widetilde{\mathbf{L}}$ is non-zero due to the finite thickness of the crystal as shown in Fig. 6.8B. This results in a separation between the entrance coordinate origin $\mathcal{O}_{in}$ and the exit coordinate origin $\mathcal{O}_{out}$.

In all cases, the total bending angle is

$$\begin{aligned}
\alpha_b &= \text{bragg\_angle\_in} + \text{bragg\_angle\_out} && \text{! Crystal, graze\_angle\_in} = 0 \\
\alpha_b &= \text{graze\_angle\_in} + \text{graze\_angle\_out} && \text{! Crystal, graze\_angle\_in} \neq 0 \\
\alpha_b &= 2 \, \text{graze\_angle} && \text{! Mirror}
\end{aligned} \tag{6.15}$$

With a mirror or Bragg diffraction, the bend angles are measured with respect to the surface plane. With Laue diffraction the bend angles are measured with respect to the line in the bend plane perpendicular to the surface.

For Laue diffraction, the user has the option of using the undiffracted beam (shown in red) as the reference trajectory.

The orientation of the exit coordinates (the local coordinates after the reflection) are only affected by the element's `ref_tilt` and bend angle parameters and is independent of all other parameters such as the radius of curvature of the surface, etc. The local z-axis of the entrance coordinates along with the z-axis of the exit coordinates define a plane which is called the element's `bend plane`. For a mirror, the graze angle is a parameter supplied by the user. For a crystal, the Bragg angles are calculated so that the reference trajectory is in the middle of the Darwin curve. Calculation of the Bragg angles for a crystal is given in Section §**??**.

6.2.4 Patch and Floor_Shift Elements Entrance to Exit Transformation

For `patch` (§**??**) and `floor_shift` (§**??**) elements, the shift in the exit end reference coordinates is given by Eqs. (6.5) and (6.6) with

$$\begin{aligned}
\mathbf{L} &= \begin{pmatrix} \text{x\_offset} \\ \text{y\_offset} \\ \text{z\_offset} \end{pmatrix} \\
\mathbf{S} &= \mathbf{R}_y(\text{x\_pitch}) \; \mathbf{R}_{-x}(\text{y\_pitch}) \; \mathbf{R}_z(\text{tilt})
\end{aligned} \tag{6.16}$$

The difference here between `patch` and `floor_shift` elements is that, with a `patch` element, the shift is relative to the exit end of the previous element while, for a `floor_shift` element, the shift is relative to the reference point on the origin element specified by the `origin_ele` parameter of the `floor_shift`.

6.2.5 Fiducial and Girder Elements Origin Shift Transformation

For `fiducial` and `girder` elements, the alignment of the reference coordinates with respect to "`origin`" coordinates is analogous to Eqs. (6.16). Explicitly:

$$\begin{aligned} \mathbf{L} &= \begin{pmatrix} \text{dx\_origin} \\ \text{dy\_origin} \\ \text{dz\_origin} \end{pmatrix} \\ \mathbf{S} &= \mathbf{R}_y(\text{dtheta\_origin})\,\mathbf{R}_{-x}(\text{dphi\_origin})\,\mathbf{R}_z(\text{dpsi\_origin}) \end{aligned} \tag{6.17}$$

6.2.6 Reflection Patch

A `Patch` (or a series of patches) that reflects the direction of the `z`-axis is called a `reflection patch`. By "reflected direction" it is meant that the dot product $\mathbf{z}_1 \cdot \mathbf{z}_2$ is negative where $\mathbf{z}_1$ is the z-axis vector at the `entrance` face and $\mathbf{z}_2$ is the z-axis vector at the `exit` face. This condition is equivalent to the condition that the associated **S** matrix (see Eq. (6.16)) satisfy:

$$S(3,3) < 0 \tag{6.18}$$

Using Eq. (6.16) gives, after some simple algebra, this condition is equivalent to

$$\cos(\text{x\_pitch})\,\cos(\text{y\_pitch}) < 0 \tag{6.19}$$

When there are a series of patches, The transformations of all the patches are concatenated together to form an effective **S** which can then be used with Eq. (6.18).

6.3 Transformation Between Machine and Element Body Coordinates

The `element body` coordinates are the coordinate system attached to an element. Without any misalignments, where "`misalignments`" are here defined to be any offset, pitch or tilt (§**??**), the `machine` coordinates (§6.1.1) and `element body` coordinates are the same. With misalignments, the transformation between `machine` and `element body` coordinates depends upon whether the local coordinate system is straight (§6.3.1) or bent (§6.3.2).

When tracking a particle through an element, the particle starts at the `nominal` (§6) upstream end of the element with the particle's position expressed in machine coordinates. Tracking from the the nominal upstream end to the actual upstream face of the element involves first transforming to element body coordinates (with $s = 0$ in the equations below) and then propagating the particle as in a field free drift space from the particle's starting position to the actual element face. Depending upon the element's orientation, this tracking may involve tracking backwards. Similarly, after a particle has been tracked through the physical element to the actual downstream face, the tracking to the nominal downstream end involves transforming to machine coordinates (using $s = L$ in the equations below) and then propagating the particle as in a field free drift space to the nominal downstream edge.

6.3.1 Straight Element Misalignment Transformation

For straight line elements, given a machine coordinate frame Λ_s with origin a distance s from the beginning of the element, misalignments will shift the coordinates to a new reference frame denoted E_s. Since misalignments are defined with respect to the middle of the element, the transformation between Λ_s and E_s is a three step process:

$$\Lambda_s \longrightarrow \Lambda_{\text{mid}} \longrightarrow E_{\text{mid}} \longrightarrow E_s \tag{6.20}$$

where Λ_{mid} and E_{mid} are the machine and element reference frames at the center of the element.

The first and last transformations from Λ_s to Λ_{mid} and from E_{mid} to E_s use Eqs. (6.5), (6.6), and (6.7) with the replacement $L \to L/2 - s$ for the first transformation and $L \to s - L/2$ for the third transformation. The middle transformation, by definition of the offset, pitch and tilt parameters is

$$\begin{aligned} \mathbf{L} &= \begin{pmatrix} \text{x\_offset} \\ \text{y\_offset} \\ \text{z\_offset} \end{pmatrix} \\ \mathbf{S} &= \mathbf{R}_y(\text{x\_pitch})\ \mathbf{R}_{-x}(\text{y\_pitch})\ \mathbf{R}_z(\text{tilt}) \end{aligned} \tag{6.21}$$

Notice that with this definition of how elements are misaligned, the position of the center of a non-bend misaligned element depends only on the offsets, and is independent of the pitches and tilt.

6.3.2 Bend Element Misalignment Transformation

For `rbend` and `sbend` elements there is no `tilt` attribute. Rather, there is the `roll` attribute and a `ref_tilt` attribute. The latter affects both the reference orbit and the bend position (§**??**). Furthermore, `ref_tilt` is calculated with respect to the coordinates at the beginning of the bend while, like straight elements, `roll`, offsets, and pitches are calculated with respect to the center of the bend. The different reference frame used for `ref_tilt` versus everything else means that five transformations are needed to get from the machine frame to the element body frame (see Eq. (6.20)). Symbolically:

$$\Lambda_s \longrightarrow \Lambda_{\text{mid}} \longrightarrow \Omega_{\text{mid}} \longrightarrow \Omega_0 \longrightarrow E_0 \longrightarrow E_s \tag{6.22}$$

The first transformation, Λ_s to Λ_{mid}, from machine coordinates at a distance s from the beginning of the element to machine coordinates at the center the bend is a rotation around the center of curvature of the bend and is given by Eqs. (6.5) and (6.6) with Eqs. (6.8) and (6.9) with the substitution $\alpha_b \to (L/2-s)/\rho$.

The second transformation Λ_{mid} to Ω_{mid} at the center of the element adds in the misalignments (Note that the coordinate frame Ω_{mid} is neither a machine frame or an element frame so hence the use of a different symbol Ω). Explicitly, the $\Lambda_{\text{mid}} \longrightarrow \Omega_{\text{mid}}$ transformation is

$$\begin{aligned} \mathbf{L} &= \mathbf{L}_{\text{off}} + [\mathbf{R}_z(\text{roll}) - \mathbf{1}]\ \mathbf{R}_z(\theta_t)\ \mathbf{R}_y(\alpha_b/2)\ \mathbf{L}_c \\ \mathbf{S} &= \mathbf{R}_y(\text{x\_pitch})\ \mathbf{R}_{-x}(\text{y\_pitch})\ \mathbf{R}_z(\text{roll}) \end{aligned} \tag{6.23}$$

where

$$\mathbf{L}_c = \begin{pmatrix} \rho(\cos(\alpha_b/2) - 1) \\ 0 \\ \rho\ \sin(\alpha_b/2) \end{pmatrix}, \qquad \mathbf{L}_{\text{off}} = \begin{pmatrix} \text{x\_offset} \\ \text{y\_offset} \\ \text{z\_offset} \end{pmatrix} \tag{6.24}$$

The reason why $\mathbf{L}$ has a different form from straight line elements is due to the fact that the axis of rotation for a `roll` is displaced from the z-axis of the coordinate system at the center of the bend (see Fig. **??**).

The third transformation from Ω_{mid} to Ω_0 is like the first transformation and rotates from the center of the bend to the beginning. Again Eqs. (6.8) and (6.9) are used with the substitution $\alpha_b \to -L/2\rho$.

The fourth transformation Ω_0 to E_0 tilts the reference frame by an amount `ref_tilt`:

$$\mathbf{L} = 0, \quad \mathbf{S} = \mathbf{R}_z(\theta_t) \tag{6.25}$$

The fifth and final transformation, E_0 to E_s, like the first and third, rotates around the center of the bend but in this case, since we are dealing with element coordinates, the `ref_tilt` is ignored. That is, Eqs. (6.8) and (6.9) are used with the substitutions $\theta_t \to 0$ and $\alpha_b \to L/\rho$.

Notice that with this definition of how elements are misaligned, the position of the center of a misaligned element depends only on the offsets and `roll`, and is independent of the pitches and tilt. Also the orientation of an element depends only on the pitches roll, and ref_tilt, and is independent of the offsets.

6.4 Phase Space Coordinates

6.4.1 Reference Particle, Reference Energy, and Reference Time

The `reference energy` and `reference time` are needed in evaluating the phase space coordinates of charged particles (§6.4.2).

All lattice elements, except for controller elements, have an associated `reference energy` energy. The reference energy at the start of a lattice's `root branch` (§1.5) is set in the lattice file by setting the reference momentum (`p0c`) or total energy (`E_tot`) using a `parameter` (§**??**) or `beginning` (§**??**) statement. For other branches, the energy at the start of the branch is set using the appropriate line parameter (§**??**) statement.

Note that the reference momentum `p0c` is actually the reference momentum times the speed of light so that the reference momentum has the same unit (eV) as the reference energy.

For most elements, the reference energy is the same as the reference energy of the proceeding element. The following elements are exceptions:

```
custom
em_field
hybrid
lcavity
patch
```

The reference energy of these elements is determined by tracking a particle (the "`reference particle`") through the element with the particle starting on the reference orbit and whose energy is equal to the reference energy. The energy of the particle at the downstream end is the reference energy of the element. Note: Tracking through an element to determine the reference energy is always done with the element turned on independent of the setting of the element's `is_on` (§**??**) parameter. Reference energy tracking is also done ignoring any orientation attributes (§**??**) and errors like `voltage_err`.

Besides the reference energy, lattice elements have an associated `reference time` which is computed, for most elements, by the time-of-flight of the `reference particle` assuming that the reference particle is following the reference orbit. Exceptions are `wiggler` elements which uses the time-of-flight of the actual undulating trajectory. [Actually what is used in the computation of the z phase space coordinate (Eq. (6.28)) is the sum of reference time deltas of the elements that a particle has passed through. It is not possible to assign a unique reference time to an element when particles are recirculating through elements as in a storage ring.]

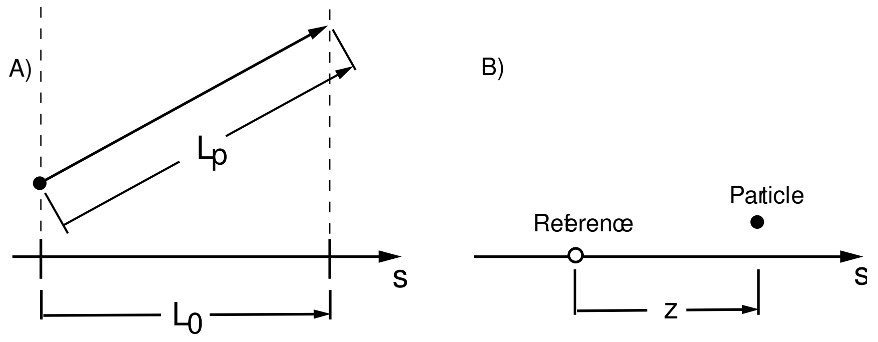

Figure 6.9: Interpreting phase space z at constant velocity: A) The change in z going through an element of length L_0 is $L_0 - L_p$. B) At constant time, z is the longitudinal distance between the reference particle and the particle.

6.4.2 Charged Particle Phase Space Coordinates

For charged particles (more correctly, for everything but photons (§6.4.4)), *Bmad* uses the canonical phase space coordinates

$$\mathbf{r}(s) = (x, p_x, y, p_y, z, p_z) \tag{6.26}$$

The longitudinal position s is the independent variable instead of the time. x and y, are the transverse coordinates of the particle as shown in Fig. 6.2A. Note that x and y are independent of the position of the reference particle.

The phase space momenta p_x and p_y are normalized by the reference (sometimes called the design) momentum P_0

$$p_x = \frac{P_x}{P_0}, \qquad p_y = \frac{P_y}{P_0} \tag{6.27}$$

where P_x and P_y are respectively the x and y momentums.

The phase space z coordinate is

$$\begin{aligned} z(s) &= -\beta(s)\, c\, (t(s) - t_0(s)) \\ &\equiv -\beta(s)\, c\, \Delta t(s) \end{aligned} \tag{6.28}$$

$t(s)$ is the time at which the particle is at position s, $t_0(s)$ is the time at which the reference particle is at position s, and β is v/c with v being the particle velocity (and not the reference velocity). The reference particle is, by definition, "synchronized" with elements whose fields are oscillating and therefore the actual fields a particle will see when traveling through such an element will depend upon the particle's phase space z. For example, the energy change of a particle traveling through an `lcavity` (§**??**) or `rfcavity` (§**??**) element is z dependent. Exception: With absolute time tracking (§**??**) fields are tied to the absolute time and not z.

If the particle's velocity is constant, and is the same as the velocity of the reference particle (for example, at high energy where $\beta = 1$ for all particles), then $\beta\, c\, t$ is just the path length. In this case, the change in z going through an element is

$$\Delta z = L_0 - L_p \tag{6.29}$$

where, as shown in Fig. 6.9A, L_0 is the path length of the reference particle (which is just the length of the element) and L_p is the path length of the particle in traversing the element. Another way of interpreting phase space z is that, at constant β, and constant time, z is the longitudinal distance between the particle and the reference particle as shown in Fig. 6.9B. with positive z indicating that the particle is ahead of the reference particle.

Do not confuse the phase space z with the z that is the particle's longitudinal coordinate in the local reference frame as shown in Fig. 6.2. By construction, this latter z is always zero.

Notice that if a particle gets an instantaneous longitudinal kick so that β is discontinuous then, from Eq. (6.28), phase space z is discontinuous even though the particle itself does not move in space. In general, from Eq. (6.28), The value of z for a particle at s_2 is related to the value of z for the particle at s_1 by

$$z_2 = \frac{\beta_2}{\beta_1} z_1 - \beta_2 \, c \, (\Delta t_2 - \Delta t_1) \tag{6.30}$$

$\Delta t_2 - \Delta t_1$ can be interpreted as the difference in transit time, between the particle and the reference particle, in going from s_1 to s_2.

The longitudinal phase space momentum p_z is given by

$$p_z = \frac{\Delta P}{P_0} \equiv \frac{P - P_0}{P_0} \tag{6.31}$$

where P is the momentum of the particle. For ultra–relativistic particles p_z can be approximated by

$$p_z = \frac{\Delta E}{E_0} \tag{6.32}$$

where E_0 is the reference energy (energy here always refers to the total energy) and $\Delta E = E - E_0$ is the deviation of the particle's energy from the reference energy. For an `Lcavity` element (§**??**) the reference momentum is *not* constant so the tracking for an `Lcavity` is not canonical.

MAD uses a different coordinate system where (z, p_z) is replaced by $(-c\Delta t, p_t)$ where $p_t \equiv \Delta E / P_0 c$. For highly relativistic particles the two coordinate systems are identical.

The relationship, between the phase space momenta and the slopes $x' \equiv dx/ds$ and $y' \equiv dy/ds$ is

$$x' = \frac{p_x}{\sqrt{(1 + p_z)^2 - p_x^2 - p_y^2}} \, (1 + g x) \tag{6.33}$$

$$y' = \frac{p_y}{\sqrt{(1 + p_z)^2 - p_x^2 - p_y^2}} \, (1 + g x) \tag{6.34}$$

$g = 1/\rho$ is the curvature function with ρ being the radius of curvature of the reference orbit and it has been assumed that the bending is in the x–z plane.

With the paraxial approximation, and in the relativistic limit, the change in z with position is

$$\frac{dz}{ds} = -g \, x - \frac{1}{2} (x'^2 + y'^2) \tag{6.35}$$

This shows that in a linac, without any bends, the z of a particle always decreases.

A particle can also have a spin. The spin is characterized by the spinor $\Psi = (\psi_1, \psi_2)^T$ where $\psi_{1,2}$ are complex numbers (§**??**).

6.4.3 Time-based Phase Space Coordinates

Some specialized routines (for example, time Runge Kutta tracking) use the time t as the independent variable for charged particle tracking. This is useful when particles can reverse direction since the normal

z based tracking cannot handle this. Direction reversal can happen, for example, with low energy "dark current" electrons that are generated at the walls of the vacuum chamber.

When the tracking is time based the phase space coordinates are:

$$(x, c\,p_x, y, c\,p_y, z, c\,p_s) \tag{6.36}$$

The positions x, y, and z are the same as with phase space coordinates (§6.4.2). The momenta are defined as

$$\begin{aligned} cp_x &\equiv mc^2\gamma\beta_x \\ cp_y &\equiv mc^2\gamma\beta_y \\ cp_s &\equiv mc^2\gamma\beta_s, \end{aligned} \tag{6.37}$$

and internally are stored in units of eV.

6.4.4 Photon Phase Space Coordinates

The phase space coordinates discussed above implicitly assume that particles are traveling longitudinally in only one direction. That is, the sign of the s component of the momentum cannot be determined from the phase space coordinates. This is generally fine for tracking high energy beams of charged particles but for photon tracking this would oftentimes be problematical. For photons, therefore, a different phase space is used:

$$(x, \beta_x, y, \beta_y, z, \beta_z) \tag{6.38}$$

Here $(\beta_x, \beta_y, \beta_z)$ is the normalized photon velocity with

$$\beta_x^2 + \beta_y^2 + \beta_z^2 = 1 \tag{6.39}$$

and (x, y, z) are the reference orbit coordinates with z being the distance from the start of the lattice element the photon is in.

In *Bmad*, the information associated with a photon include its phase space coordinates and time along with the photon energy and four parameters E_x, ϕ_x, and E_y, ϕ_y specifying the intensity and phase of the field along the x and y axes transverse to the direction of propagation. the field in the vicinity of the photon is

$$\begin{aligned} E_x(\mathbf{r}, t) &\sim E_x\, e^{i(k\,(z-z_0)-\omega\,(t-t_{\text{ref}})+\phi_x)} \\ E_y(\mathbf{r}, t) &\sim E_y\, e^{i(k\,(z-z_0)-\omega\,(t-t_{\text{ref}})+\phi_y)} \end{aligned} \tag{6.40}$$

where z_0 is the photon z position and and t_{ref} is the reference time.

The normalization between field and intensity is dependent upon the particular parameters of any given simulation and so must be determined by the program using *Bmad*.

Part III

Bibliography

Chapter 7

Bibliography

[1] Robert B. Appleby et al. “Merlin++, a flexible and feature-rich accelerator physics and particle tracking library”. In: *Comput. Phys. Commun.* 271 (2022), p. 108204. DOI: 10.1016/j.cpc.2021.108204. arXiv: 2011.04335 [physics.acc-ph].

[2] M. Borland. “elegant: A Flexible SDDS-Compliant Code for Accelerator Simulation”. In: *6th International Computational Accelerator Physics Conference (ICAP 2000)*. 2000. DOI: 10.2172/761286.

[3] H. Grote et al. “The MAD program”. In: *Proceedings of the 1989 IEEE Particle Accelerator Conference, . 'Accelerator Science and Technology*. 1989, 1292–1294 vol.2. DOI: 10.1109/PAC.1989.73426.

[4] G. Iadarola et al. “Xsuite: an integrated beam physics simulation framework”. In: (Sept. 2023). arXiv: 2310.00317 [physics.acc-ph].

[5] D. Sagan. “Bmad: A relativistic charged particle simulation library”. In: *Nucl. Instrum. Meth.* A558.1 (2006). Proceedings of the 8th International Computational Accelerator Physics Conference, pp. 356–359. ISSN: 0168-9002. DOI: https://doi.org/10.1016/j.nima.2005.11.001.