BEHAVIOR AND SCENARIO MODELING

FOR REAL-TIME VIRTUAL ENVIRONMENTS

by

Peter Jason Willemsen

An Abstract

Of a thesis submitted in partial fulfillment of the
requirements for the Doctor of Philosophy
degree in Computer Science in the Graduate College of
The University of Iowa

May 2000

Thesis supervisor: Professor Joseph K. Kearney

ABSTRACT

Virtual environments provide a powerful medium for studying human behavior. To exploit the potential of virtual environments as laboratories for psychological experimentation, we must be able to control the dynamics of complex virtual environments populated with autonomous entities. In experiments, the right things must happen at the right time and place.

My research addresses two essential components required to create engaging, real-time, virtual environments: (1) Representations of the physical environment adapted to the needs of autonomous behavior programming in urban settings, and (2) An interpreted scripting language for online programming of scenario control processes.

I introduce the Environment Description Framework (EDF) and the Scenario Description Language (SDL) as parts of an integrated system to address intertwined problems of modeling behaviors and scenarios in structured, urban environments. We create autonomous behaviors that model the dynamic entities commonly found in these environments: traffic lights, motorists, bicyclists, and pedestrians. We use the same framework to model scenario directors responsible for orchestrating the actions of multiple, autonomous behaviors.

The EDF defines a road and intersection based representation that supports programming of autonomous agents in real-time virtual environments. As its core function,

EDF gives structure and meaning to physically-based components while overlaying logical and spatial relationship information on the geometric structure of the environment. Our environment model is distinct in that it embeds behavioral constraints and inter-object relationships into the environment structure.

SDL is used for interactive modeling and prototyping of scenario directors in real-time virtual environments. SDL is an interpreted scripting language that translates scenario directives into state machines which are then inserted into the simulation execution framework. SDL advances standard scripting languages by clearly separating statements that describe what happens from statements that determine when things happen. SDL is tightly bound to simulation activity and forms a glue between the EDF, autonomous behaviors, and the scenario directors that influence the environment.

CHAPTER 4
ENVIRONMENT DESCRIPTION FRAMEWORK: AN INTERFACE TO THE WORLD

This chapter describes the *Environment Description Framework* (EDF). The EDF models road networks by defining a road-based representation that supports programming of autonomous agents in real-time virtual environments. As its core function, the EDF gives structure and meaning to physically-based components while overlaying logical and spatial relationship information on the geometric structure of the environment.

The purpose of the EDF is to efficiently model the geometric, topological, and logical information required by programs that code autonomous pedestrian and vehicle behaviors. The EDF database is also used by scenario control programs that orchestrate behaviors to create predictable experiences for participants.

The structures in the EDF are based on the nature and function of real road networks in urban environs. In the real world, driving is highly regulated; lanes channel traffic into parallel streams, signs posted alongside the road signify important changes to our behavior, and lines painted on the surface dictate whether lateral movement is prohibited, or not. Road networks are formed where roads connect to intersections. Real intersections are not free-for-alls; entrance into and movement through intersections is regulated and controlled. Intersections route incoming lanes to outgoing lanes via natural corridors. As humans, we have exceptional apparatus for perceiving these corridors and understanding the com-

plex relationships that stem from using them. Right of way conventions order movement through intersections. Traffic control mechanisms, such as traffic lights, explicitly control entrance to the intersection, often on a lane-by-lane basis. All in all, movement over roads and intersections in everyday life is controlled by well-defined structures, societal rules, and automated regulation mechanisms to minimize congestion and accidents.

The organization of the EDF is highly influenced by the behaviors used to control autonomous agents. In this regard, the EDF is *behavior-centric*. The key to this organization comes from an understanding of what information, in addition to geometry and topology, is needed by the autonomous agents in the environment. Through this organization, the EDF emphasizes a strong coupling between behavior and environmental structure. In essence, behavioral implications are furnished by the environmental structure to assist autonomous agent decision making processes. The coupling between behavior and environment is central to ecological psychology. J.J. Gibson introduced the idea of environmental affordances for behavior. He asserted that humans perceive much more from the environment than basic sensory information. He argued that the environment provides humans with contextual and relational information [28][56]. In a similar sense, most man-made structures, such as roads and intersections, are designed to administer a specific logic and order to our behavior. I believe that these principles and design considerations serve as excellent motivation and reference for constructing both the environmental framework and the interactive, autonomous behaviors within the virtual environment.

The EDF defines structural, logical, and relational properties in the database com-

ponents to integrate behavior and environment representations. Structural components provide detailed geometric and topologic information. For instance, roads and intersections represent structural components. The road surface is composed of purely geometric representation forming the so-called pavement of the environment. Intersections have shape and define a boundary. Roads connect to intersections along this boundary creating the road network topology.

Logical attributes describe the culturally important and socially responsible aspects of the urban environment. For the most part, these properties influence the autonomous agents' actions within the confines of the EDF's structural components. Lanes, lane lines, roadside features, intersection corridors, and traffic control mechanisms all represent logical aspects of the EDF.

In the EDF, relational components describe spatial and inter-object adjacency information. In addition, relational components are used to link important behavioral relationships with context-dependent behavior. Occupancy information is calculated by the simulation and stored in the EDF to spatially locate the autonomous agents with respect to the structural components of the EDF. Adjacency is defined in terms of which objects are in front of, behind, or next to the nearby autonomous agents. Corridor dependencies mark important intersection relationships on a corridor-by-corridor basis.

The treatment of intersections in the EDF illustrates one aspect of how these properties combine to form the EDF's components. Intersections are notoriously difficult to model, both geometrically and behaviorally, in driving simulations. Roads interconnect in

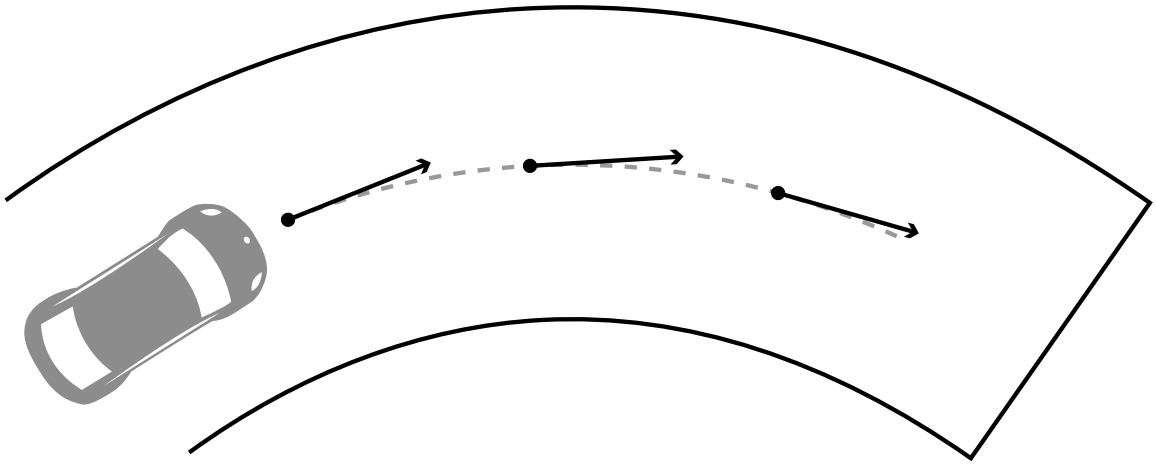

Figure 4.1: A vehicle navigates over a ribbon-like surface inquiring about the tangent and curvature properties at various points along the surface.

complicated ways; driving behaviors must understand where they can go, how they can get there, and how their behavior relates to pedestrians and other vehicles whose paths cross their own. Behaviors must also have access to traffic control devices whose time-varying states govern acceptable behavior in the intersection. The EDF database integrates the structural and logical properties of the intersection with information about the relational interdependencies among objects simultaneously traveling through the intersection.

4.1 Curvilinear Coordinates

Road-like structures can be modeled as ribbons in space. A roadway is naturally expressed in terms of a space curve with surface orientation defined by surface normals. Objects in the simulation environment follow the curvature of the road. Figure 4.1 illustrates a vehicle following the curvature of a ribbon-like surface. Objects avoid collisions by detecting surrounding objects and taking appropriate actions to avoid contact. For instance,

vehicles stay within the lanes of roads and slow down to avoid collisions with objects in front of them. Pedestrians dodge and weave around nearby pedestrians on crowded sidewalks.

The driving surface in the EDF is described by a reference curve in 3D space with a surface extending outward, away from the curve. Way points, features, and object positions are all located relative to this surface. Elevation, curvature, tangent, and surface normals are important surface or curve features that are embedded in the surface description. The values of these attributes may change as a function of surface position. For example, curvature at one point on the surface may not be the same as curvature just a small distance away. In general, the information needed by autonomous behaviors is often dependent on structural context and location on the road surface.

Road surfaces in the EDF are represented in *curvilinear coordinate systems*. Curvilinear coordinates provide a natural means for describing positions and relative locations on the road surface. A curvilinear coordinate system is a two-dimensional coordinate system based on a three-dimensional reference curve that acts as the major axis. The minor axis is defined as the vector perpendicular to the major axis and the surface normal.

The curvilinear coordinate system used in the EDF defines the first dimension as *distance* (δ) and the second dimension as *offset* (o). Distance is measured as the arc-length along the reference curve. Offset is given by the shortest distance to the reference curve as measured linearly along the minor axis. A single curvilinear coordinate is described by the pair (δ, o). Using this system, points on the surface can be uniquely located with both a

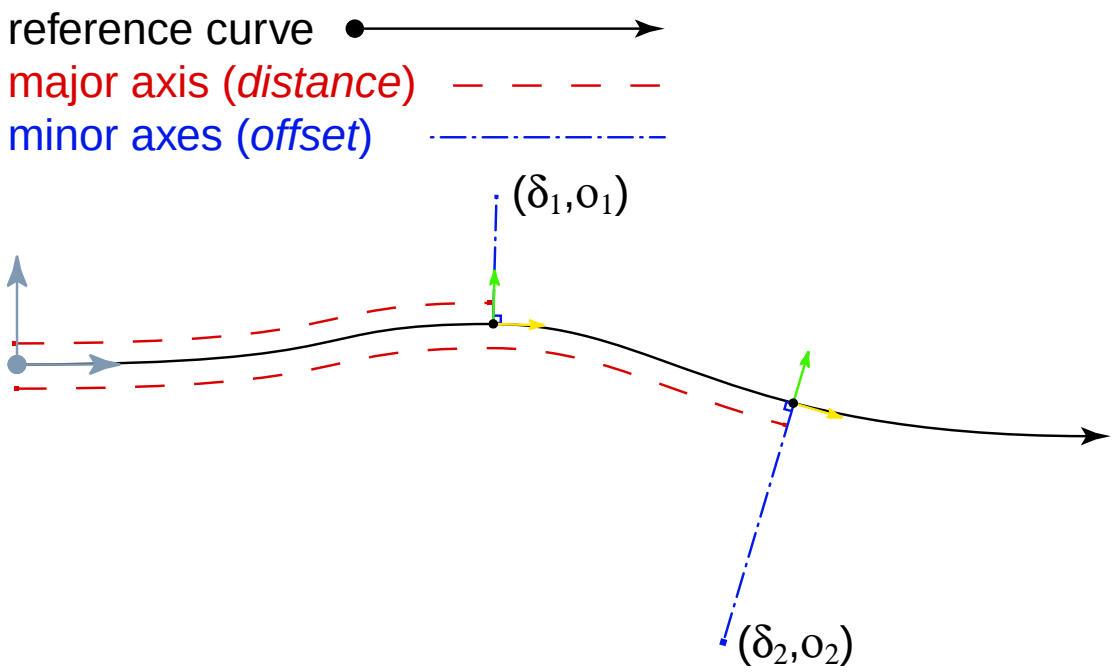

Figure 4.2: Illustration of a curvilinear coordinate system. Two curvilinear coordinates are shown depicting different locations.

distance (δ) and an offset (o) parameter.

Coordinates on a curvilinear surface are defined from an origin, which is defined as the starting location of the reference curve. At this location, both distance and offset are zero. Offsets with positive value are located to the left of the reference curve and offsets with negative value are located to the right (when viewed in the positive direction of the major axis). A curvilinear coordinate frame is illustrated in Figure 4.2. The distance and offset measures define ordering relations for objects on the surface that are the basis for determining spatial relationships of objects on roads in the EDF.

Curvilinear coordinates are used in the EDF database as a basis representation for

all navigable routes. Behavioral objects, such as vehicles and pedestrians, track curvilinear coordinates to follow roads, lanes, and routes through intersections. In Chapter 3, I mentioned that one of the potential responsibilities of the autonomous behaviors is to provide control inputs to physics-based, dynamics models of the objects they control. At times it is beneficial for behaviors to compute control inputs in terms that are consistent with the dynamics models controlling the object. Most dynamics models are expressed in terms of Cartesian coordinates, and thus, produce Cartesian positions, rather than curvilinear positions. Therefore, it is necessary to provide conversions between curvilinear coordinates and Cartesian coordinates. These conversions are an essential part of all curvilinear-based components in the EDF.

Given a (δ, o) pair, any curvilinear coordinate system in the EDF database needs to be capable of transforming the coordinate into a Cartesian coordinate, as expressed by the EDF query

$$\vec{R^3} \Longleftarrow queryXY((\delta, o))$$

This query takes as input a curvilinear coordinate and returns a Cartesian coordinate in three-space.

Once the dynamics process has calculated a new Cartesian position, we must map the position back to curvilinear coordinates. In EDF, the transformation function from Cartesian to curvilinear coordinates is

$$(\delta, o) \Longleftarrow queryDO(\vec{R^3})$$

which converts the input parameter R^3 into a curvilinear coordinate (δ, o) on the curvilinear

coordinate system requesting the transformation.

The ability to convert coordinates from Cartesian to curvilinear space is important at other times as well. When objects are initially placed in the simulation environment, it is often easier to position the objects according to absolute Cartesian coordinates rather than curvilinear coordinates. Once in the environment, the object's location can be converted to curvilinear coordinates using the provided queries, as needed.

For road-based behaviors, curvilinear coordinates are extremely useful: (1) they naturally represent locations along a road-like surface thus facilitating road and lane tracking behaviors, (2) relative spatial locations are easily defined with respect to the curve which is instrumental for following and obstacle avoidance behaviors, and (3) conversions between curvilinear and Cartesian coordinates afford flexibility in programming autonomous agents as different coordinate systems can be used for different purposes.

4.2 Segments - The Pavement Under Our Feet

In the real world, when construction crews build new roadways they first grade the road and then lay pavement where the new roads, or intersections, will exist. At this stage in construction, only the surface geometry is defined. In the EDF, drivable surface geometry is represented by basic structures called *segments*.

EDF segments are analogous to the pavement of real road networks and are used to model fixed width stretches of road. EDF segments are represented with curvilinear coordinate systems to provide geometric and structural information about the surface. While segment width is constant, the extent need not be symmetric about the curvilinear refer-

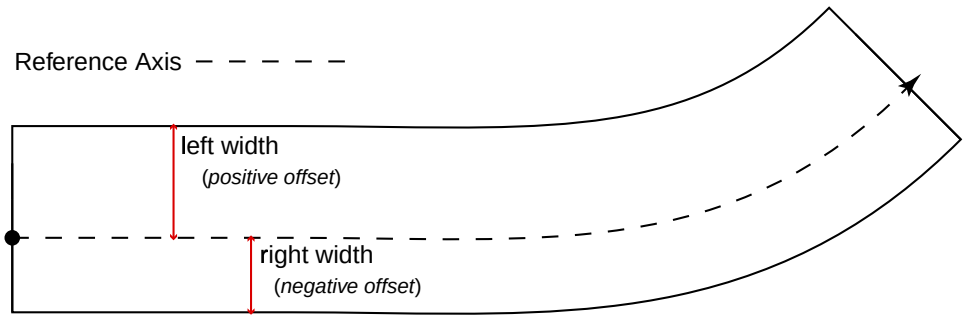

Figure 4.3: Illustration of general segment properties.

ence axis. Figure 4.3 illustrates a basic EDF segment. Segments are structural components that support curvilinear coordinate transformations, as well as geometric queries about the curvilinear system and surface, including curvature, tangent, elevation, and surface normal.

Using a variety of segments, we are able to create road-like structures in the EDF on which vehicles and pedestrians can drive and walk. For instance, we can construct linear segments for representing straight roads and cubic segments for representing the surface descriptions necessary for curved paths. In general, we can construct many different ribbon-like segment descriptions each for representing different road surfaces. For the most part, the roads we model do not twist violently or turn upside down. However, because the surface is defined by the reference curve's normal vectors, EDF segments can easily be used to represent surfaces that exhibit extreme twist, such as can be found on roller coaster tracks. Nonetheless, our concentration is on urban driving environments where, for the most part, the road normals point skyward.

Because the segment is based on a curvilinear system, surface attributes, such as

float $\Longleftarrow$ *length()* Returns the total arc-length of the segment.
float $\Longleftarrow$ *width([left or right])* Returns the left or right width of the segment.
float $\Longleftarrow$ *elevation(* (δ, o) *)* Calculates surface elevation at (δ, o) on the segment.
float $\Longleftarrow$ *curvature(* (δ, o) *)* Calculates curvature at (δ, o) on the segment.
$\vec{R^3}$ $\Longleftarrow$ *tangent(* (δ, o) *)* Calculates the tangent vector at (δ, o) on the segment.
$\vec{R^3}$ $\Longleftarrow$ *normal(* (δ, o) *)* Calculates the surface normal vector at (δ, o) on the segment.
(δ, o) $\Longleftarrow$ *queryXY(* $\vec{R^3}$ *)* Converts from Cartesian to curvilinear coordinates.
$\vec{R^3}$ $\Longleftarrow$ *queryDO(* (δ, o) *)* Converts from curvilinear to Cartesian coordinates.

Table 4.1: The queries supported by segments in the EDF. All segments must support these queries. Queries are often parameterized by curvilinear coordinates to access exact surface detail.

elevation, tangent, curvature, or surface normal can be queried as functions of (δ, o) coordinates. Such curvilinear data must be well-defined for each segment. Table 4.1 lists and briefly describes the behaviorally important queries provided by EDF segments. Necessary parameters and return types are also noted.

These queries are designed to be used in a real-time simulation system. Therefore, it is highly important that the segment queries and the mappings between curvilinear and Cartesian coordinates be efficient, fast, and robust. Many thousands of queries may occur each second. In this context, a segment implementation that requires a complex integration calculation in order to transform from one of the coordinate systems to the other may

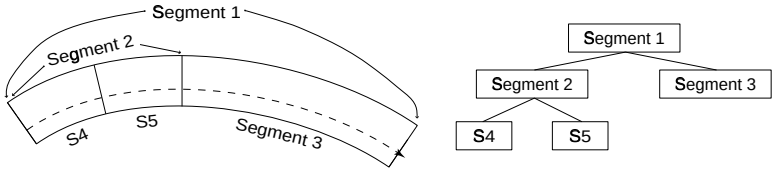

Figure 4.4: An illustration of a hierarchical segment in the EDF. The complete segment is composed of three leaf segments [*S4*, *S5*, *Segment3*]. Segments *Segment1* and *Segment2* are parent segments.

be too computationally expensive. In any case, it is important that speed, efficiency, and robustness be balanced for segment types used in the EDF.

4.2.1 Compositing - Hierarchical Segment Descriptions

It is useful to combine segment definitions to form larger aggregates. Segments in the EDF can be composed of other segments forming hierarchical curvilinear surface descriptions. Parent segments assemble and manage child segments to transparently present a single EDF segment interface. In other words, a single curvilinear coordinate system is mapped across a the leaf segments. Figure 4.4 illustrates the hierarchical structure for a simple segment composed of three leaf segments.

Hierarchical segments are continuous at the joins of adjacent segments. With regard to the connection between adjacent segments, we define continuous to mean that (1) connecting segments' end points be coincident, (2) adjacent segments' widths are identical on both positive and negative sides, (3) adjacent segments' reference curves align with at least C^1 continuity[1], and (4) surface normals at the joining edge of the two segments are

[1] C^1 continuity requires a common tangent at the joining point of the two segments and C^2 continuity requires that the curvature be common at the joining point of the two segments.

identical. These restrictions ensure that hierarchical segments remain ribbon like in curve and surface definitions.

4.2.2 Segments in the EDF

EDF segments are used to model both geo-specific and geo-typical roadways. In the real-world, road designers construct modern highways using standard techniques of civil engineering. However, many older roads are not built to conform to any standard design criteria. Using the EDF, we can model many different types of roadways using civil engineering data, or more free-form open-ended design criteria.

To match civil engineering design standards, the EDF defines three segments based on analytic expressions used in roadway design. These are the *straight*, *curve*, and *spiral* segments [3, 23]. These segments are described analytically and model smoothly changing curvature and super elevation required on high-speed roadways. When composited together in the EDF these segments can be used to accurately model many real-world roads. Figure 4.5 illustrates how a curved road would be represented in the EDF using a combination of these segments. Because of their analytic nature, these segments afford fast and efficient queries for surface characteristic information and curvilinear coordinate system transformations.

4.2.2.1 Straight Segments

As seen from an orthographic projection onto the ground plane, straight segments in the EDF connect two Cartesian coordinates in space with a line. Surface elevation, and thus,

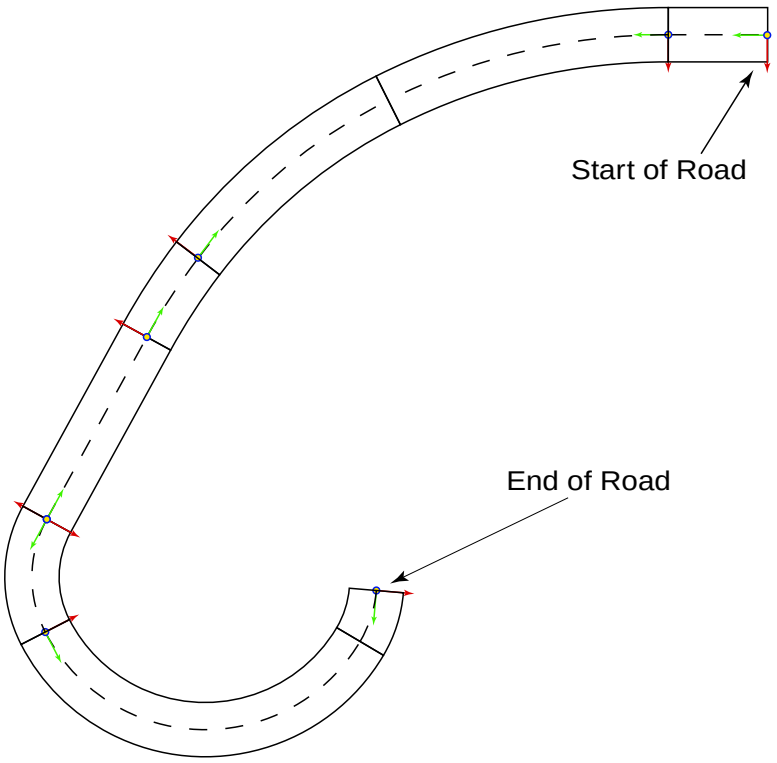

Figure 4.5: A longer and more complex road can be built by compositing segments. In this example, only three different types of segments are used: straight, curve, and spiral.

surface normal, is controlled by a separate function of δ that bases height on a parabolic curve. In the EDF, straight segments can be specified with either two, or three, Cartesian coordinates. When two coordinates are defined, the segment surface is flat, with linear changes in X, Y, and Z coordinates. When specified using three coordinates, the straight segment's elevation (height in Z) is determined by fitting a parabola through the three Z coordinates. Two-dimensional tangent vectors along a straight segment are constant, while curvature is always zero.

4.2.2.2 Curve Segments

Curve segments define a ribbon-like circular arc with Z dependence identical to the straight segment. In other words, a curve segment surface has constant circular radius (constant curvature) and sweeps out an arc of constant radius. If only the two end coordinates of the arc are specified than the change in the elevation (Z) is linear with respect to length along the curve. By specifying a third coordinate, we can fit a parabola to the segment describing surface elevation as a function of distance along the curve. Curve segments may also define a constant *super-elevation* which specifies the amount of banking. Surface elevation at a point is determined as by combining the surface height with the super-elevation.

4.2.2.3 Spiral Segments

Spiral segments provide a transition between straight and curve segments. Spiral segments smoothly interpolate between the zero curvature and zero super-elevation of a straight segment to the fixed curvature and super-elevation of a curved segment. The eleva-

tion of the spiral segment is determined by combining super elevation with surface height. The surface height of the spiral is calculated identically to the straight and curve segments with either a linear interpolated profile or a parabolic elevation profile running the length of the segment.

4.2.2.4 Generalized Cubic Segments

At times, it is more convenient and easier to model roads in a free-form manner than by adjoining analytical segments. Free-form road construction is useful for quickly prototyping scenes using intuitive curve drawing techniques. Moreover, many roads in the real world are not constructed according to standard engineering design practices. In the EDF, free-form road modeling is handled by a generalized *cubic* segment. A cubic segment is described by a set of n interpolating points in R^3.

$$\{p_0, p_1, \ldots, p_{n-1}\}$$

To facilitate transformations between curvilinear and Cartesian coordinate systems, we must reparameterize the cubic segment so that it is parameterized by arc-length. Once the segment is parameterized by arc-length, curvilinear coordinates can efficiently be converted to Cartesian coordinates by performing binary searches using curvilinear distance as the key over the sampling points, calculating localized cubic evaluations. Determining curvilinear coordinates from Cartesian coordinate can also be difficult. We use use a small number of steps in a Newton's method to efficiently handle conversion from Cartesian to curvilinear coordinates.

In summary, EDF segments provide the foundation for modeling road surfaces.

EDF segments are curvilinear coordinate surfaces that analogous to ribbons of pavement through the environment. Segments provide queries for interrogating the surface information and converting between local curvilinear coordinates and Cartesian space. Using hierarchical composition, segments can be combined to form more elaborate pathways that mimic the construction style of roads found in the real world.

4.3 Roads

Once the pavement for a new road is set, construction crews paint lane lines on the road surface that delineate lane structure. Lanes lines also identify regulations on driving activities, such as passing. Signs are placed alongside the road to signify speed limit, no parking, merges, and yields. The lane lines and signage represent logical aspects of driving. Humans can easily drive over pavement, but order, and thus behavior, is determined from the logic applied to the road. In the EDF, roads are modeled very similarly by applying logic and order to a segment representing the pavement.

Roads in the EDF are composed of a segment, which may be hierarchical, a lane profile definition, and feature descriptions. The road's segment definition represents the pavement on which the road logic will be applied. EDF roads connect to intersections at the termination points of the roads. Through the intersection, other roads can be reached.

Roads in EDF derive their width from their segment. The road's width is then laterally subdivided into parallel channels called *lanes*. Lanes are classified according to behavioral use and run parallel to the curvilinear axis defined by the road's segment description. Lane definitions are specified outward from the road's axis to ensure consistent and easily

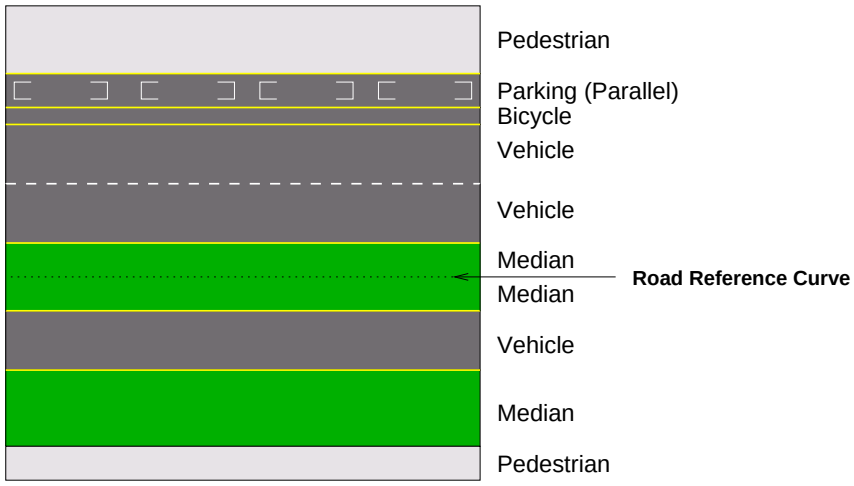

Figure 4.6: Road width is subdivided laterally across the width of the road to form lanes. Lanes can be of varying types and in this example sidewalks, vehicle lanes, bicycle lanes, parking lanes, and medians are shown.

identifiable lane boundary offsets that are based on the lanes' widths. Lane boundaries are used to define adjacency relationships between neighboring lanes. The treatment of lane definitions in the EDF is depicted in Figure 4.6, which shows a cross-section of a lane definition.

As a person drives over the length of a real road, that person's behavior is apt to change and reflect the features and markers posted or painted along the roadway. Some types of markings, such as speed limit zones, affect large expanses of road while others, BUMP or LANE ENDS signs for instance, provide more immediate and local information. In the EDF, *range attributes* describe curvilinear expanses of road in which specific behavioral or societal rules apply. Similarly, the EDF uses *features* to model localized, situated information that exists at specific distances along the length of the road.

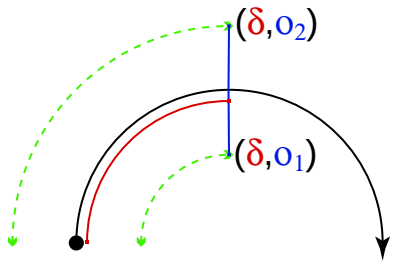

Figure 4.7: Curvilinear distance is calculated along the reference curve (shown in black). Both coordinates have the same distance parameter (δ), but different offsets (o_1, o_2). Dashed, green arcs represent the true arc-length of the curve at the different offsets.

During a simulation, the EDF maintains information about the simulation objects located on each road's surface at any instance of time. This information can be queried providing behaviors with important road occupancy detail. Object location on roads, as well as how those objects' locations relate to each other, facilitates following and obstacle avoidance behaviors. A complete discussion of road occupancy is provided in Section 4.5.

4.3.1 Interpretation of Curvilinear Coordinates

Distance along the length of a road in the EDF is treated similar to mile-markers along an interstate in the real world. A mile-marker marks an invisible perpendicular line that cuts across all lanes of a highway to denote the location of a particular mile, or distance from the beginning of the roadway. Any vehicles crossing that line, no matter what lane they are in, are considered to be located at that mile marker.

By our definition, curvilinear distance (δ) is measured along the reference curve; curvilinear offset (o) is calculated as the shortest perpendicular distance to the reference curve at that distance. This interpretation of a curvilinear coordinate uses a single longitu-

dinal representation of distance, as opposed to an offset based definition of distance over the curve. Thus, two positions (δ, o_1) and (δ, o_2) define a line perpendicular to the road reference curve at milemarker δ. Figure 4.7 illustrates the how the EDF treats curvilinear coordinates in terms of mile markers.

The advantage of using this interpretation for the curvilinear coordinate is that important spatial relationships among objects are directly determined. For instance, given a set of of distance parameters on a curvilinear surface, we can define an ordering of those distances over length of the surface. Similarly, using a set of offset parameters, we can determine an ordering that runs laterally across the surface definition. Using these partial orderings, autonomous behaviors can query neighboring objects and compare curvilinear distance coordinates to understand local spatial relationships.

4.3.2 Lanes

Lanes partition the road surface into parallel, longitudinal regions. A road in the EDF bundles a set of lanes together. Roads can be composed of one or more lanes, with the lane set being ordered and dense–meaning that there are no gaps between adjacent lanes. Lane widths are constant over the length of the road. As a consequence, the road profile is constant over the length of the road as determined by the extents of the lane offsets.

In the EDF, a road's lanes run parallel to the road's curvilinear axis and may not straddle that axis. Thus, the road axis is always used as a boundary between two lanes. This restriction is beneficial because it allows for a discrete and simple ordering of lanes. Consequently, a curvilinear offset can quickly be mapped to a particular lane by utilizing

Lane Type	Description
VEHICLE	General-purpose vehicles
BICYCLE	Bicyclists
SIDEWALK	Pedestrians
BUS	Buses
TRAIN	Train tracks
TRAM	Trolley cars
AGRICULTURAL	Tractors, combines, threshers, wagons
CENTER-TURN	Specialized turn lane
MEDIAN	Lane spacing
PARKWAY	Area between road pavement and sidewalk
ANGLED PARKING	Angled parking lanes
PERPENDICULAR PARKING	Perpendicular parking lanes
PARALLEL PARKING	Parallel parking lanes

Table 4.2: Various types of lanes defined in the EDF.

the lane width and adjacency information.

Lanes in the EDF are typed. For instance, a *sidewalk* is a type of lane dedicated to pedestrians while a *center-turn* lane allows traffic to make left turns on busy streets without blocking traffic flow. We chose to associate sidewalks with roads because sidewalks, and other similar features, fit nicely with our definition of lanes. Generally, sidewalks accompany roads in the real world, running parallel to the road for the length of the road. Table 4.2 lists and briefly annotates the lane types in EDF. In addition to behavioral type information, lanes designate a preferred direction of flow, oriented with respect to the road coordinate system. Some lanes allow bidirectional movement, so flow direction in a lane may take on one of three values: `Positive`, `Negative`, or `Both`. Each lane can specify a surface-height offset value which is used to raise, or lower, the lane's elevation relative

to the height of the its road. This attribute is useful for medians, parkways, and sidewalks which are sometimes raised above the height of the road to which they are adjacent.

4.3.3 Range Attributes

In the real-world, signs and markings are posted alongside and on roads to highlight important information related to driving. For instance, speed limits signs indicate the maximum speed allowed on a section of road. Lanes lines are drawn between lanes to prohibit or allow certain types of behavior, such as passing in a passing zone.

The EDF models zone-based, road characteristics as *range attributes*. Range attributes are bounded curvilinear regions that supply behaviorally important information about that region of the road surface. Range attributes are characterized by a rectangular curvilinear area and data. A range attribute's area is defined by a pair of curvilinear coordinates $\{(\delta_{ll}, o_{ll}), (\delta_{ur}, o_{ur})\}$ specifying the lower left (δ_{ll}, o_{ll}) and upper right (δ_{ur}, o_{ur}) coordinate of road surface. The range attribute's data is represented by an attribute label, followed by a list of data elements

$$(attribute\_label, data_0, data_1, \ldots, data_{n-1})$$

In the case of a speedlimit range attribute, the label is *speedlimit* and the data associated with it would be the speed limit over that region, as in

$$(speedlimit, 65mph)$$

During the course of a simulation, range attributes can be dynamically controlled by an autonomous behavior. Consider the following, a railroad crossing gate can be rep-

resented as a dynamically manipulated range attribute. In this example, a region of the road is marked to denote the area of the road where the train tracks cross. The bounds of this region demarcate the location of the gates that guard access to the crossing area. The crossing gate attribute data is specified by

$$(rrcrossing, \delta_1, PositiveFlowGate, up, \delta_2, NegativeFlowGate, up)$$

The curvilinear distances (δ_1, δ_2) given in the range attribute data component determine the location of the respective gates along the road surface. The two gates, labeled as *PositiveFlowGate* and *NegativeFlowGate* to signify the relative direction of traffic flow they toward which they face, guard access to the railroad track crossing region. The logical state of these gates is either *Up* or *Down* to represent the physical state of the gate. By attaching an autonomous behavior to this range attribute, we provide a mechanism to dynamically monitor the crossing train tracks for an oncoming train and set the state of the range attribute gates appropriately if a train is detected. Autonomous behaviors, such as vehicles, navigating the road will query the *rrcrossing* range attribute and can react to the logical state of the gate, either stopping prior to the gates, or driving over the road normally.

4.3.4 Features

When driving on real roads, signs posted along the roadway will, at times, denote immediate and local information specific to that place on the road. For instance, a SPEED BUMP sign often marks the location of a speed bump on the road's surface. In the EDF, road signs, such as these, that mark a perpendicular reference on a road surface are represented as *features*.

A feature is a logical EDF component that models localized, cross-sectional information on the road's surface. Features have a facing direction and can be laterally bounded by two offsets bracketing where the feature resides on the road. Features differ from range attributes in that they do not cover an area on the road; rather, features cover a cross-sectional slice of the road surface. A feature's placement on the road's surface is determined by a distance δ, two offsets bounding the left and right offsets that the feature affects (o_{left}, o_{right}), and a facing parameter, γ.

$$\text{feature location } = (\delta, o_{left}, o_{right}, \gamma)$$

Feature facing direction determines the direction toward which the feature points. For instance, on real two-way streets, signs for the oncoming lanes face toward the oncoming traffic and do not affect the opposite flow traffic lanes. Features that are POSITIVE are directed toward traffic traveling toward the end of the road's curvilinear coordinate system; NEGATIVE facing features are directed toward traffic moving toward the origin of the road's curvilinear coordinate system. Features can face both directions.

The data component of a feature is represented by a feature label followed by a series of data elements.

$$\text{feature data } = (feature\_label, data_0, data_1, \ldots, data_{n-1})$$

Like range attributes, feature data may be dynamically controlled by an autonomous behavior over the course of a simulation.

The EDF supplies queries for interrogating range attributes and features on roads. Range attributes occur over curvilinear regions of the road, and thus, queries that are pa-

list<RangeAttr> $\Longleftarrow$ queryRangeAttributes((δ, o)) Returns the list of range attributes that span the given curvilinear coordinate.
list<Feature> $\Longleftarrow$ queryFeatures($(\delta, o), (\delta', o'), \gamma$) Return a list of features between the bounding curvilinear coordinates, given the facing direction.
RangeAttr $\Longleftarrow$ queryRangeAttribute(attribute_label, (δ, o)) Returns the requested range attribute that exists at the given curvilinear coordinate.
Feature $\Longleftarrow$ queryFeature(feature_label, $(\delta, o), (\delta', o'), \gamma$) Returns the requested feature between the bounding curvilinear coordinates, given the facing direction.
list<RangeAttr> $\Longleftarrow$ queryRangeAttributesByName(attribute_label) Returns a list of range attributes on a road with the specified attribute label.
list<Feature> $\Longleftarrow$ queryFeaturesByName(feature_label, γ) Returns a list of features on a road with the specified feature label, given the facing direction.

Table 4.3: EDF queries that support interrogation of range attributes and features on roads.

rameterized by (δ, o) return all the range attributes affecting those particular curvilinear coordinates. On the other hand, because features have facing directions and are specified cross-sectionally at exact distances along the road surface, EDF queries restrict the search for features with a bounded curvilinear region and facing direction. Table 4.3 lists the EDF road queries used to query information about range attributes and features.

4.3.5 Additional Queries for Interrogating Road Information

Roads in EDF use the curvilinear coordinate system provided by its segment. Thus, roads provide the same queries as listed for the segments (Section 4.2). Additional queries provide topologic information and inform behaviors about the intersections to which the

int $\Leftarrow$ *numberOfLanes()* Returns the number of lanes on the road.
list<Lane> $\Leftarrow$ *laneProfile()* Returns the set of lanes that make up the lane profile on the road.
Lane $\Leftarrow$ *laneToLeft(lane_reference)* Returns the lane to the left (if it exists) of the lane parameter.
Lane $\Leftarrow$ *laneToRight(lane_reference)* Returns the lane to the right (if it exists) of the lane parameter.
Intersection $\Leftarrow$ *startIntersection()* Returns a reference to the road's starting intersection.
Intersection $\Leftarrow$ *endIntersection()* Returns a reference to the road's ending intersection.
Intersection $\Leftarrow$ *nextIntersection(flow_direction)* Given a direction of flow on the road, return the forward facing (as determined by flow direction) intersection.
float $\Leftarrow$ *distanceToIntersection(* δ*, [start* $\mid$ *end])* Given a curvilinear coordinate distance parameter and an intersection label, return the distance to the intersection.
float $\Leftarrow$ *distanceToNextIntersection(* δ *, flow_direction)* Given a curvilinear coordinate distance parameter and a flow direction, return the distance to the forward facing intersection.

Table 4.4: Additional queries supported by roads in the EDF.

road connects. Relative queries support queries based on the behavior's immediate direction of flow with respect to the road's curvilinear coordinate system.

4.4 Intersections

In the real world, intersections serve as interchanges between different roads. Driving behavior is different in intersections than on roads. Roads are simply organized into parallel streams in which human behavior can be tightly regimented, and reasonably predictable. Intersections, on the other hand, are logically complicated, messy, and, in the field of driving simulation, notoriously difficult to model. The channel-like passages connecting incoming to outgoing lanes criss-cross and overlap each other. Spatial relationships become tangled, highly complex, and difficult to identify. In real intersections, societal rules and conventions determine who has right of way and how traffic should proceed.

In the EDF, intersections specify input/output relationships between the lanes of the roads connected to the intersection. EDF intersections define a standard interconnection scheme that details how lanes interlink and interrelate in the intersection. Intersections in the EDF have four parts: (1) an exterior polygonal boundary to which roads incident to the intersection connect, (2) sets of individual, lane-like corridors that route traffic through the intersection, (3) lists of dependencies between corridors that describe how objects on different corridors relate, and (4) traffic control devices that regulate entrance into and movement through the intersection.

4.4.1 Intersection Boundaries - Exterior Geometry

Intersections in the EDF are bounded by a convex polygon. The edges of the polygon are used as landmarks to which the terminating edges of connecting roads can adjoin, forming smooth transitions between road and intersection geometry.

As with roads, simulation objects can occupy intersections. The convex geometric description of the intersection polygon allows the EDF to perform efficient object in polygon computations during a simulation. Intersection occupancy semantics will be discussed in Section 4.5.

4.4.2 Corridors

In real intersections, the connections between lanes are often implicitly defined. At times however, especially in the presence of adjacent turning lanes, or when it isn't visually obvious, lanes lines are painted in the intersection to explicitly guide drivers on how they should drive through the intersection. In the EDF, the connections between lanes are explicitly represented and describe the routes between incoming and outgoing lanes of roads that connect to the intersection. These routes are called *corridors*.

Corridors are analogous to single-lane roads. They contain a segment, which may be hierarchical, that describes a curvilinear path between the two lanes incident on the intersection. The corridor's reference curve connects the termination points of two lane centerlines on the intersection boundary. Thus, the corridor's reference curve is a lane-centerline to lane-centerline route. Corridor width is determined to be the maximum of the two connecting lanes' widths. Figure 4.8 illustrates a simple intersection with corridors for

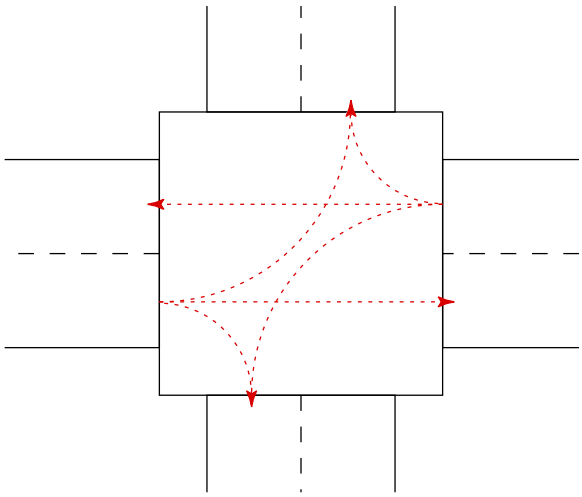

Figure 4.8: Corridors cross intersections and connect incoming lanes to outgoing lanes. The corridors leaving the left and right roads are shown in red.

the left and right roads drawn in dashed lines. Corridors can represent any route through the intersection. The figure shows commonly used routes for driving straight, turning right, and turning left.

Corridors are typed. Corridor type information serves the same purpose as lane type does for EDF roads. Both provide contextual information to behaviors. The EDF places no restrictions on which corridor types can match up with which lane types. This means that logically incompatible connections can be constructed and it is up to the modeler to correctly match compatible types. For instance, a bike lane can reasonably connect to a vehicle lane, but connecting a vehicle lane to a median, sidewalk, or tram lane is not sensible. In the former, both the bike lane and vehicle lanes support drivable vehicles that maneuver on paved surfaces. In the latter, it is not clear how a vehicle would drive on a tram track, let alone a sidewalk, or median.

4.4.2.1 Corridor Attributes - Flow, Stoplines, and Curbs

At the intersection boundary, incoming lanes connect to corridors leading to lanes on outgoing roads. When more than one corridor branches out from the same incoming lane, multiple pathways exist through the intersection from that lane. Behaviorally, these junctures are noteworthy because they force behaviors to make decisions as to which corridor to use. Often times, random decisions will suffice.

Unfortunately though, random decisions produce random behavior that may not be consistent with the design of the environment. From the standpoint of the database modeler, these junctures present an opportune place to encode modeling intentions and design notes that reflect the modeler's expectations about traffic movement on the lanes and corridors of the environment.

Similarly, traffic modeling algorithms can use corridor branches to encode desired traffic flow statistics that will influence turn decisions of autonomous vehicles. Algorithms for managing ambient traffic and maintaining various traffic models attempt to control traffic density and flow on roadways by influencing the routing decisions of traffic in intersections [2, 12, 11].

The EDF encodes a *flow percentage* attribute in each corridor indicating the desired percentage of traffic entering the intersection on that lane that should take the given corridor. The accumulated percentages for all corridors leaving a lane must sum to one. EDF enforces this constraint by accepting flow percentages between 0 and 1 and then normalizing the set of flow percentages to be consistent over the range $[0, 1]$. For corridors that are

bidirectional, flow percentages are specified at both ends of the corridor.

By encoding flow percentage into the corridor, autonomous behaviors, such as vehicles or pedestrians, can make educated, yet random decisions that are consistent with desired traffic distributions. Moreover, database and traffic modelers can prescribe flow percentages that match the design of the road, or scenario being developed without recoding the vehicles that drive in the environment.

Flow percentages can be dynamic to evolve to changes in simulation state. For instance, autonomous traffic manager behaviors can modify and tweak the flow percentages at intersections during a simulation to route traffic according to time varying scenario requirements and traffic distribution rules and goals.

On roads in the United States, most controlled intersections are marked with stoplines to designate a stopping location for vehicles. The EDF encodes *stopline distance* for each corridor that is used to enter the intersection. This is primarily useful if the intersection boundary encompasses the crosswalk and stopline markings. Otherwise, stopline can be encoded as a feature along the roadway. Stopline distance is a distance measured along the curvilinear reference curve of the corridor and can be queried by behaviors approaching the intersection to ensure that the object being controlled by the behavior stops at the appropriate place along the corridor. In situations where multiple corridors branch from a single incoming lane, it may be the case that stopline distance will be different for each corridor.

Stoplines are primarily for vehicles and mark an important location on the road

surface. For pedestrians, curbs act like a stoplines, guarding entrance into the roads. More importantly, curbs often signify abrupt change of elevation. In the EDF, it is important that pedestrian behaviors have access to the location of the curb so that foot falls can accurately, and realistically be placed on the curb [65]. The EDF models the location of curbs by specifying a *curb distance* attribute on pedestrian corridors. Curb distance is encoded into the corridor as the distance from either the beginning, or the end, of the corridor. As with stopline distance, curb distance on bidirectional corridors is specified at both ends, if required.

Flow percentage, stopline distance, and curb distance are specialized attributes built into the corridor description. In addition to these specialized, built-in attributes, corridors, like roads (see Section 4.3), can specify range attributes, and/or features along the curvilinear coordinate system used to define the corridor.

4.4.2.2 Junctures

The intersection boundary uniquely defines the geometry of the intersection. Roads abut to the edges of the intersection making the transition from road to intersection smooth and seamless. Similarly, internal corridors seamlessly align with the intersection boundary. Lanes on connecting roads map to specific corridors in the intersection. In general, there are many attributes and elements that need to be specified; intersections are difficult to model.

Luckily, in real urban environments multiple intersections often have the same relative structure and characteristics, with only minor differences between intersections being

the widths of the roads that connect to the intersections. Downtown city block intersections are an excellent case in point. In the synthetic environment, reuse of structure is even more widely used and therefore apparent in the models.

Intersections are difficult to specify and model. In the EDF, we leverage modeling effort by affording intersection reuse. It makes sense to reuse the structure, logic, and relational characteristics defined in the intersection, especially when the only differences between intersections may be the incoming lane widths.

The EDF facilitates intersection reuse by defining connection sites called *junctures*. Junctures are placed along the edges of the intersection boundary and mark locations to which roads, lane centerlines, and corridors can connect. By pre-defining these connection points, intersections can be constructed independent of their connections to roads. As a consequence, corridors can be specified without regard to which roads or lanes they connect. In this sense, intersection definitions becomes templates that can be instantiated as needed in the environment description. Upon instantiation in the EDF, the bindings between the connecting roads, lanes, and corridors become effective and explicit, cementing the intersection at that location.

Two types of junctures are used in the EDF: *fixed* and *floating*. Fixed junctures mark an exact location on the intersection boundary edge. They are useful for connecting roads and well-defined corridors to the intersection. Floating junctures describe a connection point on the intersection boundary that is not yet bound to a specific location on the edge. Floating junctures are used when we know a connection will be made, but we don’t know

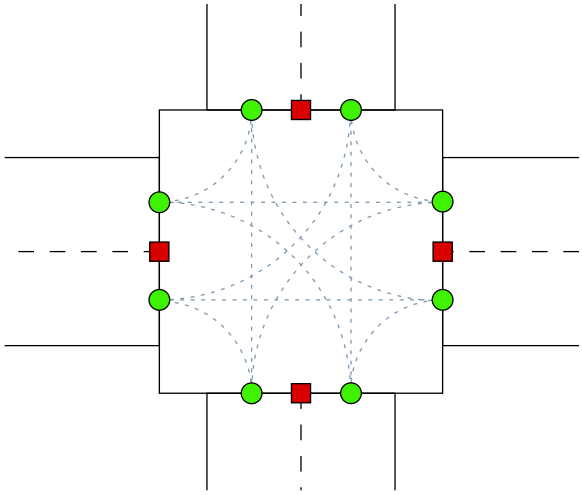

Figure 4.9: Illustration of fixed and floating junctures placed along the intersection boundary. Red squares denote fixed junctures while green dots represent floating junctures. The corridors between the junctures are shown as well.

at what point on the boundary edge the connection will exist. Floating junctures are the primary tools for making intersections reusable. Internally, corridors can be connected to both fixed and floating junctures.

Multiple junctures can be placed on an edge to accommodate multiple connections. When more than one juncture is placed along an edge, the ordering of junctures is always preserved by the EDF. Figure 4.9 illustrates a simple intersection with both types of junctures. The red squares represent fixed junctures, while the green dots represent floating junctures.

Juncture ordering along an edge is invariant. When initially defined, fixed junctures remain fixed along the edge while floating junctures are spread out evenly between neighboring absolute junctures and edge vertices.

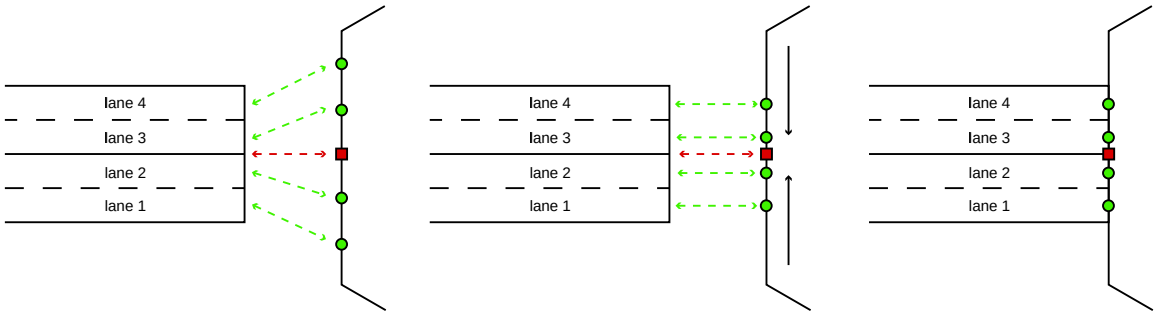

Figure 4.10: The road axis connects to the fixed juncture (shown as a red square) and the floating junctures (shown as green circles) adapt to fit the offsets of the matching lanes. This three-part illustration shows the process of connecting a road to an intersection.

When we connect a road to an intersection, we attach the road's reference axis to a fixed juncture on the intersection boundary. Only the terminating coordinates of roads may connect to a juncture. However, more than one road may connect to an edge as long as it connects to a separate fixed juncture.

Once a road is connected to an intersection, the EDF attempts to align the neighboring floating junctures with the road's lanes' centerlines. Floating junctures connect to the ending, or beginning, coordinates of lane centerlines. Floating junctures translate along the boundary edge to accommodate for the fixed spacing of the lanes on the roads. A floating juncture pushes, or pulls, neighboring floating junctures as it translates along the edge to accommodate the width of the connecting lane. Juncture ordering is always preserved along an edge. Figure 4.10 illustrates the process of connecting a road to an intersection and how the junctures adapt to fit the road's lanes.

To assist modeling effort, the EDF can generate cubic spline segments to model the corridor geometry between junctures automatically. These automatically defined segments

are beneficial when using floating junctures. They allow the EDF to adapt the corridor definition to the locations of the floating junctures once roads and intersections connect.

On the other hand, corridor segments can be defined explicitly, but in these situations, the EDF cannot adapt the corridor segment to fit updated locations of floating juncture endpoints. It is suggested that in these circumstances, fixed junctures be used to ensure rigid corridor placement.

In terms of database modeling, junctures allow intersections to be modular and reusable. A database modeler can reuse an intersection description by delaying the binding, or connection, between road and juncture until a later time. When designing and developing a database model, a single intersection can be instantiated as needed, connecting roads and lanes to the appropriate junctures on the intersection edge. In essence, this produces intersection descriptions that are parameterized by the set of roads that connect to the intersection.

4.4.3 Dependencies - Internal Intersection Constraints

As corridors pass through intersections they interweave in complex ways: crossing, overlapping, merging, or splaying. Many corridors feed into the same outgoing lane and single incoming lanes diverge into many corridors connected to different outgoing lanes. In real intersections, traffic control devices and right of way conventions govern interactions of vehicles on different corridors. Without these, chaos would ensue.

As drivers enter and negotiate travel through intersections, they need to attend to many differing concerns and constraints. For instance, turning left is preceded by locating

Dependencies
ADJACENT_TO
CROSSES
CROSSWALK
DIVERGES_WITH
MERGES_WITH
RIGHT_ON_RED
RIGHT_OF_WAY

Table 4.5: Types of dependencies that can be used in the EDF to describe the inter-object relationships on corridors.

a safe gap in oncoming traffic. Even driving straight through an intersection requires that the driver pay some attention to the traffic on crossing corridors.

The constraints that are implicitly defined between the objects traveling through intersections are corridor dependent. The EDF encapsulates inter-object constraints into corridor *dependency* information. Dependencies define potentially hazardous and important interrelations among traffic on different corridors. The EDF encodes dependencies into the intersection on a per-corridor basis. Simply put, dependencies supply behaviors with context dependent information related to the corridor on which they are navigating. A dependency maps a dependency label to one or more corridors, as in

$$(dependency\_label, corridor_0, corridor_1, \ldots, corridor_{n-1})$$

Multiple dependencies can be specified for each corridor in an intersection and are encoded into the environment by the database modeler depending upon the capabilities of the autonomous behaviors. The EDF does not specify a preset list of dependencies for use by behaviors. To provide a feel for different types of dependencies, Table 4.5 lists a set of

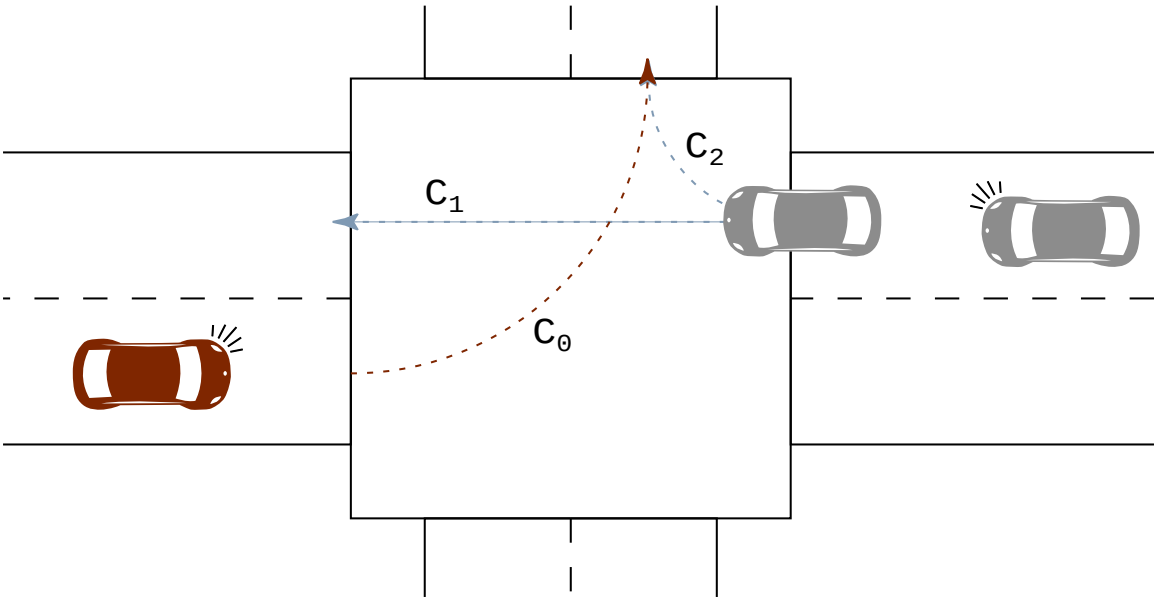

Figure 4.11: An illustration of the use of dependencies to negotiate a left turn. Vehicles traveling on corridor C_1 and C_2 may pose threats to objects traveling on corridor C_0.

common dependencies.

Dependencies compartmentalize behavioral activities. Each dependency in the intersection signifies a relationship between the objects on two corridors. Autonomous behaviors can be designed and organized around these relationships, so that as dependencies are encountered, separate, behavioral-dependency units can be deployed to react to the dependency on an as-needed basis.

For example, assume an autonomous vehicle needs to make a left turn at an intersection, as depicted in Figure 4.11. The darker vehicle will enter the intersection on the corridor labeled C_0. In this circumstance, the vehicle's turning behavior must accept a safe gap in the oncoming traffic. The oncoming traffic corridors denoted by C_1 and C_2 are

specified in the dependency list of corridor C_0 as

$$(\textbf{merges\_with}, C_1)$$

$$(\textbf{crosses}, C_2)$$

The darker vehicle's behavior can use the dependency list to dynamically create two turning dependent sub-behaviors: one for dealing with the merging traffic, and another for accepting a safe gap in the oncoming traffic. After the turn is completed, these two newly instanced behaviors can be retired until later use.

4.4.4 Traffic Control

Behavior in EDF intersections is regulated by traffic control devices (*e.g.* stop signs, traffic lights, pedestrian walk signals) that control traffic flow on a per-corridor basis. Bidirectional corridors, such as sidewalks, are controlled unilaterally by a single traffic control state. Autonomous vehicles, for instance, can interrogate the intersection for the traffic control state associated with the corridor the vehicle plans to use to travel through the intersection.

Traffic control state may be static or may be determined dynamically by an autonomous behavior. For example, in the EDF, a single traffic light behavior can control the state of a particular intersection's traffic light. The state of the traffic lights at other intersections are usually controlled by separate behaviors. However, at times, all the traffic lights at a series of intersections may be synchronized with each other to maximize traffic flow through the road networks.

Traffic control state can be determined by multiple behaviors. In uncontrolled inter-

Traffic Control States
RED
YELLOW
GREEN
FLASHING_RED
FLASHING_YELLOW
STOP_SIGN
THROUGH
UNCONTROLLED

Table 4.6: Traffic control state.

sections, no traffic control device is required. Table 4.6 lists the traffic control states used in the EDF to regulate behavior at intersections.

4.4.5 Zero Length Intersections

Polygonal intersections specify a fixed geometry with corridors mapping routes between incoming and outgoing lanes. However, at times, it makes sense to simply connect the ends of two roads together without the behavioral, and modeling, overhead associated with a complete intersection.

In essence, the intersection structure is collapsed to a single edge–a degenerate polygon. In the EDF, these intersections are called *zero-length intersections* since they occupy no space yet act like regular, polygonal-based intersections.

Zero-length intersections use *zero-length corridors* to describe lane-to-lane connections and specify dependency information between corridors. Zero-length corridors have no length, and are point-based connections that provide a mapping between lanes connect-

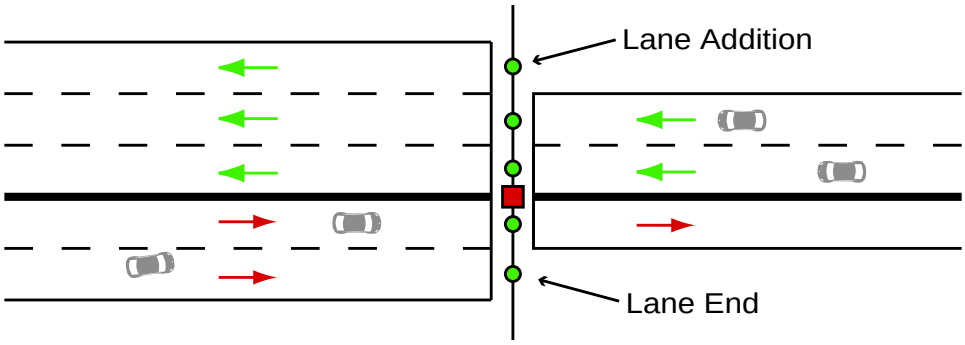

Figure 4.12: Example of how a zero-length intersection connects two roads with differing lane definitions.

ing to the junctures on the zero-length intersection's edge.

Traffic control at zero-length intersections works identically to traffic control in regular EDF-based intersections. However, zero-length intersections are used to describe seamless changes in road structure where, for the most part, no traffic control state is required and traffic can drive through without change of behavior.

There are two instances where zero-length intersection are used: (1) forming road loops, and (2) lane termination or lane addition. Figure 4.12 depicts an instance of using a zero-length intersection to connect two roads with differing lane definitions. When using zero-length intersections, lane centerlines must align on the connecting roads since there are no corridors to merge between lanes of different offsets.

4.4.6 Hierarchical Intersections - Internal Routing

The EDF intersection model uses hierarchical intersection descriptions to provide the necessary structure for describing complex internal intersection routing. To understand

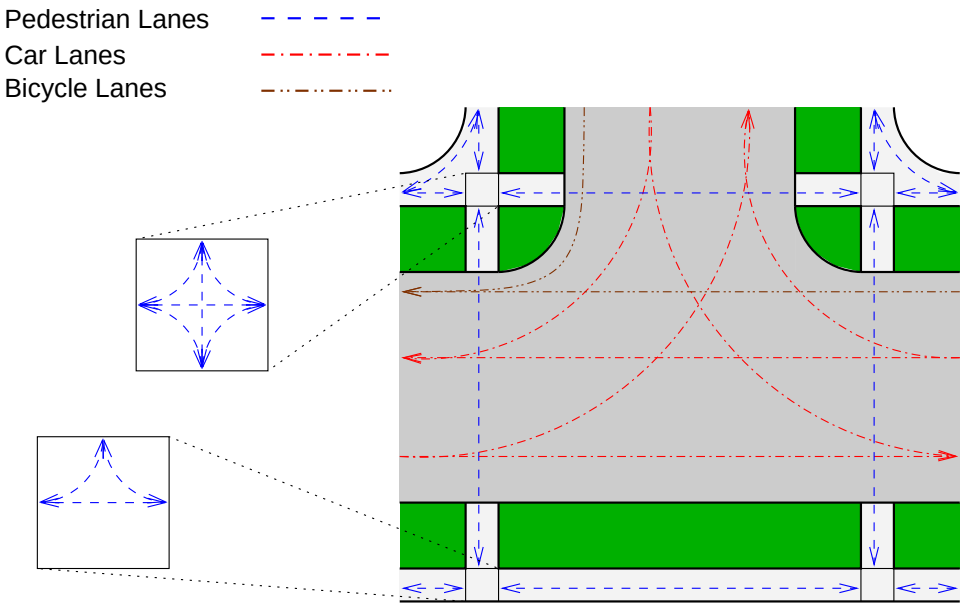

Figure 4.13: Example of a hierarchical intersection definition. Note the internal intersections forming the pedestrian sidewalk intersections.

internal routing, consider the manner in which sidewalks intersect. Sidewalk lanes in the EDF connect to sidewalk corridors. These corridors criss-cross forming their own intersections. Often times, the location of these pedestrian intersections is within the bounds of the intersection geometry. Within internal intersections, corridors connect to corridors at junctures on the edges of the child intersection. Figure 4.13 illustrates a three-way intersection where sidewalks come together within the boundary of the larger intersection. At these crossings, we define internal intersections.

4.4.7 Intersection and Corridor Behavior Queries

Table 4.7 lists queries provided by the intersection and corridor structures in the EDF. Intersection queries provide access to corridors connecting roads, lanes, and corridors

Intersection Queries
list<Corridor> ⇐= queryCorridors(Road, Lane) Returns a list of Corridors that connect to the road and lane provided as parameters.
list<Corridor> ⇐= queryCorridors(Corridor, [start | end]) Returns a list of Corridors that connect to the specified end of the corridor provided as parameter if the intersection is an internal intersection.
TCState ⇐= queryTCState(Corridor) Returns the traffic control state value regulating movement along the specified corridor.
boolean ⇐= isZeroLength() Returns a boolean value indicating if the intersection is a zero length intersection.
Corridor Specific Queries
list<Dependency> ⇐= queryDependencies() Returns a list of dependencies on the corridor.
boolean ⇐= connectsToRoad([start | end]) Returns a boolean value indicating whether the specified end of the corridor connects to a road, or not. If false, the corridor connects to an internal intersection.
Intersection ⇐= startIntersection() Returns a reference to the intersection at the beginning of the corridor.
Intersection ⇐= endIntersection() Returns a reference to the intersection at the end of the corridor.
Intersection ⇐= nextIntersection(flow_direction) Given a relative direction of flow on the corridor based on the corridor's curvilinear axis, return the forward facing (as determined by flow direction) intersection.
float ⇐= distanceToIntersection(δ, [start | end]) Given a curvilinear coordinate distance parameter and an intersection label, return the distance to the intersection.
float ⇐= distanceToNextIntersection(δ , flow_direction) Given a curvilinear coordinate distance parameter and a flow direction, return the distance to the forward facing intersection.

Table 4.7: Intersection queries relate connecting roads and corridors and provide traffic control state for corridors. Corridor queries provide access to dependency information and which intersections (in the case of internal intersections) connect to the corridor termination points.

together and also give access to the traffic control device state at the intersection. Because corridors are curvilinear structures, they support all the queries that a curvilinear coordinate system must provide, as described in Section 4.2. They also support the feature and range attribute queries described in the road section (Section 4.3). In addition, EDF corridors provide access to dependency and internal intersection information.

4.5 Occupancy

In addition to the structure and layout of the static environment, it is essential that dynamic objects be aware of other dynamic objects. For example, a vehicle on a road must avoid crashing into the vehicle immediately ahead of it in its lane. Pedestrians dodge and weave around other pedestrians on the sidewalk. To be able to avoid collisions and to interact in other ways, autonomous characters need access to the dynamic objects in the environment (*e.g.* vehicles, pedestrians), but more importantly, they need access to the spatial locations of the objects near them in the environment.

The EDF maintains relative and spatial locations of objects. Relative locations refer to the relationships between objects and the surfaces in the environment on which they exist. In other words, relative locations describe *occupancy* on a surface. For example, relative location information can provide answers to queries, such as what vehicles are on a particular lane, what pedestrians are in a crosswalk, or what objects are in an intersection.

Spatial location simply describes the location, or position, of the object in the environment. This can be expressed in two ways. For one, spatial location can be the object's Cartesian coordinates. On the other hand, if the object is located on a curvilinear surface,

spatial location can be a curvilinear coordinate, described by δ, o.

Occupancy is derived from the spatial positions of objects and how they relate to the structures of the simulated environment. But first, we need an understanding of what it means to occupy something.

4.5.1 Occupying Space in the EDF

In the EDF, the term *occupancy* means how simulation objects occupy space on roads, intersections, and corridors. For our purposes, occupancy is a by-product of an orthographic projection of the objects' geometry onto the ground surface, and hence onto the EDF structures covering the ground. It is quite important that both animate and inanimate objects be included in occupancy computation. For instance, we expect pedestrians walking on the sidewalk to dodge and weave around each other to avoid collisions. Likewise, we expect similar behavior if there are lamp posts or parking meters placed along the sidewalk.

The answer to the question *Does a particular object occupy a particular structure?* depends upon who asked the question and why the question was asked. Some behaviors simply may not require as much detail about occupancy as others. For instance, whether or not a vehicle's bumper is encroaching into another vehicle's lane depends upon the attitude of the driver. A conservative behavior may find the movement of the neighboring vehicle's bumper offensive and dangerous, while an inattentive driver may not even notice the bumper.

Nonetheless, it is important to establish a baseline for occupancy. At the most primitive level, each object in the simulation is represented by a single point denoting the

position of it's center of geometry (CoG). This coordinate can be used for assigning point-based occupancy.

4.5.2 A Witness to Occupancy

We formulate descriptions of occupancy in terms of *witnesses*. This is in contrast to describing occupancy in terms of the simulation objects themselves. As an example, we can describe occupancy on a road by listing all the objects on that road. The witness description is an occupancy abstraction that can tell us what objects occupy a certain structure, but can also provide additional information that describes how an object occupies a space. For instance, a point-based witness provides the most primitive form of occupancy in the EDF by mapping an object's CoG onto one or more EDF components[2]. This witness would provide a reference to the occupying object as well as the point of occupancy.

EDF does not restrict occupancy to being a point-based, CoG computation. The witness metaphor is an intuitive abstraction for expressing occupancy and is advantageous in that it allows for many different types of occupancy representations. A witness in the EDF attests to the time and placement of a specific object on a particular road, intersection, or list of corridors. A witness specifies a description (geometric or otherwise) regarding how the object occupied space on the EDF component. This includes a reference to the occupied object. For instance, our point-based witness provides an autonomous behavior with the following information: (1) a reference to the occupying simulation object, (2) a time-stamp dating the occupancy, (3) the Cartesian coordinate, and one of the following:

[2]Since roads and intersections do not allow for overlap, it is only in the presence of overlapping corridors that a CoG may be on more than one component at a time.

Road Queries
list<Witness> ⇐= queryObjects() Returns all point-based witnesses to road's occupants
list<Witness> ⇐= queryObjectsInLane(lane_ref) Returns all point-based witnesses to occupants on a particular lane
Witness ⇐= queryLeader(distance, ori) Returns a witness representing the next object in the direction specified by *ori* after *distance* on a road
Witness ⇐= queryLeaderInLane(lane_ref, distance, ori) Returns a witness to the leader occupant within a lane
list<Witness> ⇐= queryAllLeadersInLane(lane_ref, distance, ori) Returns a list of witness to all the objects ahead of a particular distance
Intersection Queries
list<Witness> ⇐= queryObjects() Returns all point-based witnesses to intersection occupants
list<Witness> ⇐= queryObjectsInCorridor(Corridor, width) On demand query that calculates the occupants within a specified width of the corridor axis
Witness ⇐= queryLeaderInCorridor(Corridor, width, distance, ori) On demand query that calculates the leader ahead of a given distance on a corridor
General EDF Occupancy Queries
list<Object> ⇐= queryAllObjects() Returns all the objects in the simulation

Table 4.8: Primitive point-based witness queries provided by the EDF. These queries use point-based witnesses for occupancy computations.

(a) a reference to a road plus a curvilinear coordinate, (b) an intersection reference, or (c) a list of corridor references plus curvilinear coordinates.

During every simulation step, we compute point-based witnesses for each dynamic simulation object, but for roads and intersections only. Corridor occupancy is computed on demand, as needed by behaviors. The reason for this is that most of the time, objects entering intersections are only concerned with the occupants on a small number of corri-

dors. The intersection dependency structures support this style of corridor occupancy by modularizing the inter-object relationships on the corridors of the intersection; behaviors need only focus on the objects on specific corridors. Thus, given a particular corridor and intersection, the EDF can compare the CoG positions of the objects in the corridor with the single corridor. Contrast this computation with the calculations required to compute point-based occupancy on all corridors in the intersection. In this situation, each corridor is compared with each object. These computations are more often than not, unnecessary and wasteful. Therefore, we leave it to behavior programs to request corridor occupancy as needed. Table 4.8 lists the primitive point-based queries provided by the EDF. As an example, we can query all the objects on a road by using the function *queryObjects* with a road reference. This function would return back a list of point-based witnesses whose CoG map onto a projection of the road surface.

4.5.3 Sophisticated Witnesses

The EDF leverages the definition of occupancy on to the witness abstraction and computation. Point-based occupancy works fine for some behaviors, such as lane based following, but it may not be sufficient for more complex parallel parking behaviors, or lane encroachment detection behaviors. Moreover, the point-based witness is imprecise; it doesn't tell the behavior much about *how* the object is situated at its position.

More intelligent, informed, and sophisticated witnesses can be built to supply detailed information about how objects occupy structures in the environment. Consider a line-based witness that represents geometry by one or more line segments, describing how

each line segment relates to structures in the EDF. A line-based witness can be used to represent the extent of an object in one direction. The witness may output a list of line segments accompanied by the structures those line segments overlay. For instance,

$$\text{line witness} = \{object\_ref, (l_1 : [x_1, y_1], [x_2, y_2], road : r1, lane : 1), \\ (l_2 : [x_2, y_2], [x_3, y_3], road : r1, lane : 2)\}$$

which lists two line segments (l_1 and l_2) and the structures on which they are positioned. The line segments represent portions of the object referred to by *object_ref* such as a line representing the longitudinal extent of the object and how it may be positioned on the underlying surface. In this example, the object may be changing lanes as part of it is located on lane 1 while the other line segment is located on lane 2. Similarly, an area-based witness may partition the convex hull of a simulation object into polygonal subsets, describing how each sub-region occupies different space on adjacent lanes, roads, or intersections.

4.5.4 Level of Detail Occupancy Queries

Often times, the distances between objects is a good indicator for how much occupancy information may be important to a behavior. Nearby objects raise more immediate concerns than do far away objects. For instance, a behavior that avoids collisions with other objects in a lane is more concerned with the objects immediately in front of it than it is with the objects two miles down the road.

To facilitate efficient access to occupancy information for behaviors, the EDF performs on-demand occupancy witness calculations with level of detail (LOD) thresholds to manage computation expense. LOD occupancy calculations allow us to concentrate

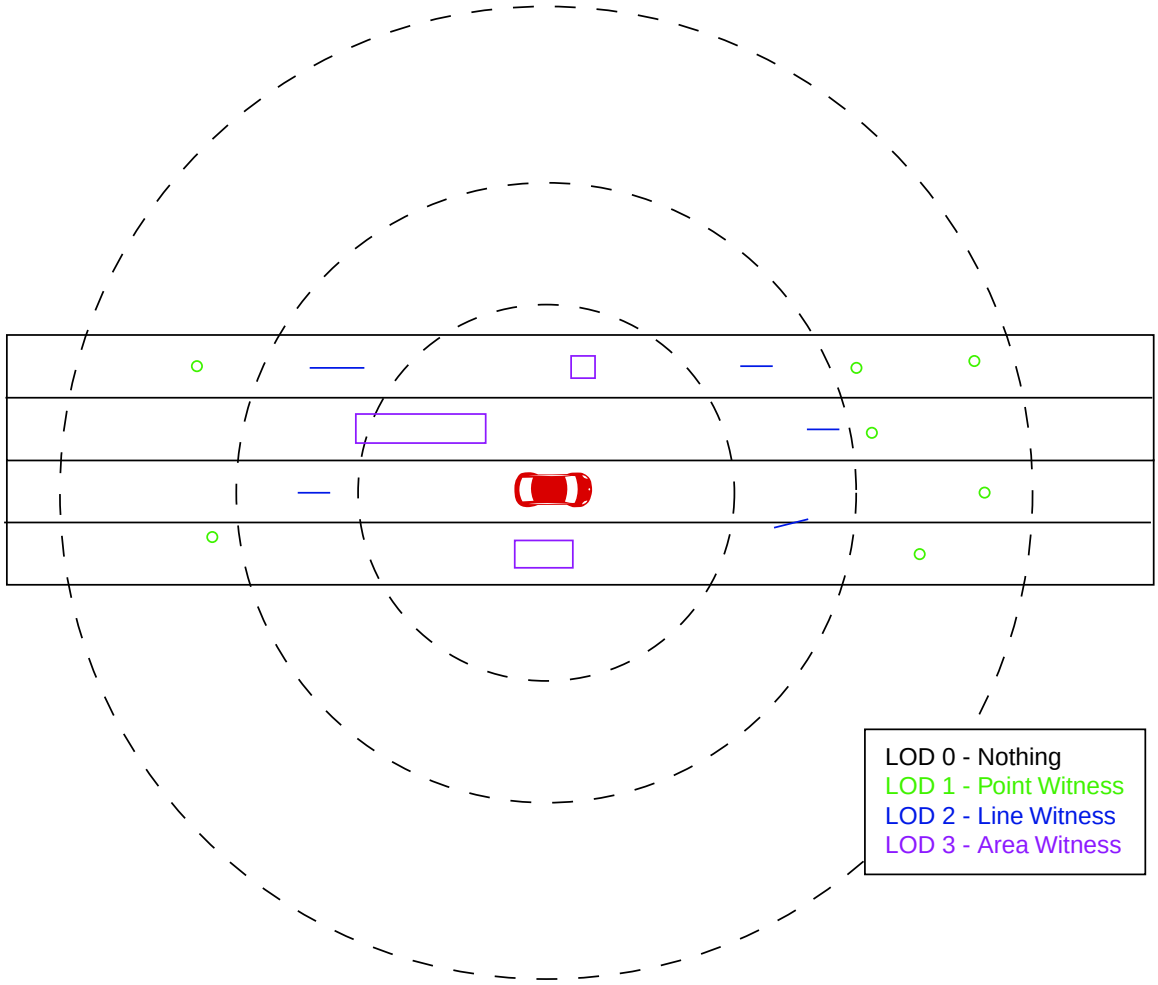

Figure 4.14: Level of Detail (LOD) occupancy queries provide general solution to different occupancy needs. Witnesses at different LODs provide different detail.

computation where behaviors need it. Figure 4.14 depicts an autonomous behavior and a visualization of the results from querying LOD occupancy. Four levels of detail are shown. In the example, witness calculations closest to the vehicle provide area-based occupancy information (highest level of detail) and witness calculations farther from the vehicle return point-based occupancy status (lowest level of detail). The objects farthest from the vehicle (outside the outer circle) are not returned as a result of this query.

Witness calculations for the higher level LODs are computed on demand and are uniquely described and implemented for each simulation object. For instance, consider the difference between pedestrian and vehicle witnesses. Pedestrian occupancy at the highest level of detail may utilize a bounding circle computation whereas the highest level of detail in a vehicle may perform calculations based on how the convex hull of the vehicle occupies EDF structures. Level of detail calculations are described for each type of object to take advantage of the particular object's geometric definition.

Behavioral objects specify which level of detail occupancy witnesses they require. This is accomplished by providing the occupancy queries of the EDF with a LOD profile stating which LOD witnesses to compute at different distance thresholds. A LOD profile is a list of distance and LOD pairs. For example,

$$\text{LODProfile} = \{(d_0, HighLOD), (d_1, MediumLOD), (d_2, LowLOD)\}$$

which specifies that the highest LOD witness calculations should be returned for simulation objects that are less than or equal d_0 distance away from the behavior requesting the information. The medium LOD should be used for objects between d_0 and d_1 units away,

Road Queries
list<Witness> $\Longleftarrow$ LOD_queryObjects(distance, LODProfile) Returns road occupancy witnesses based on the LODProfile and the distance parameters
list<Witness> $\Longleftarrow$ LOD_queryObjectsInLane(lane_ref, distance, LODProfile) Returns lane occupancy witnesses based on the LODProfile and distance parameters
Witness $\Longleftarrow$ LOD_queryLeader(distance, ori, LODProfile) Returns a witness based on the LODProfile that represents the next object in the direction specified by *ori* after a particular *distance* along the road
Witness $\Longleftarrow$ LOD_queryLeaderInLane(lane_ref, distance, ori, LODProfile) Returns a witness based on the LODProfile representing the leader occupant within a lane
Intersection Queries
list<Witness> $\Longleftarrow$ LOD_queryObjects($\vec{R}^3$, LODProfile) Returns witnesses based on the LODProfile and the distance between the intersection occupants and the Cartesian vector provided as a parameter
list<Witness> $\Longleftarrow$ LOD_queryObjectsInCorridor(Corridor, distance, width, LODProfile) On demand query that calculates the occupants within a specified width of the corridor axis using the LODProfile and distance along the corridor
Witness $\Longleftarrow$ queryLeaderInCorridor(Corridor, width, distance, ori, LODProfile) On demand query that calculates the leader witness ahead of a given distance on a corridor using the LODProfile

Table 4.9: Level of detail (LOD) witness queries for EDF roads, intersections, and corridors. LOD occupancy witness queries are computed on an as-needed basis.

and the lowest LOD calculation should be performed for objects between d_1 and d_2 units away. Simulation objects beyond d_2 units away are not returned as witnesses. Table 4.9 lists the different level of detail (LOD) witness occupancy queries provided by EDF for roads, intersections, and corridors.

4.6 Paths

Throughout this chapter, I have used the word *path* as a generic description for a navigable route used by simulation objects. In the EDF, the word *path* has more specific meaning denoting an abstraction designed to support the continuous and smooth navigation of objects over the road networks. For the remainder of this chapter the word *path* will be used as it pertains to this restricted meaning.

A path provides a means for tracking and following objects on a lane-based route over sections of the road networks. Paths are intended to be used for local navigation, providing immediate information about the route and the occupants along that route. Paths are composed of an ordered sequence of (road, lane) pairs and (intersection, corridor) pairs. Paths manage their aggregate parts to present a single lane-based curvilinear coordinate system. Paths do for lanes and corridors what roads do for lists of segments (see Section 4.3).

The path abstraction is a design decision influenced purely by behavior modeling issues dealing with intersections. With regards to vehicle behaviors, intersections are difficult to navigate. This is even more so for pedestrian behaviors using hierarchical intersections. A single path eliminates the managerial tasks required by a behavior to negotiate complex

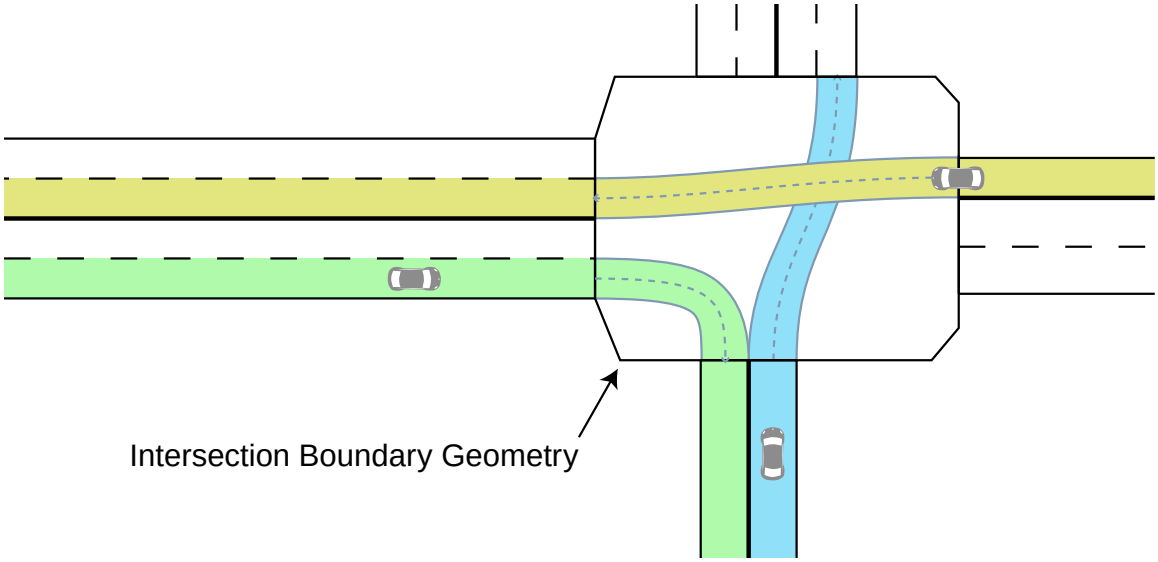

Figure 4.15: Illustration of path use by vehicles. Three paths are shown for three different vehicles. The paths cross into and through the intersection boundary.

intersections, and allows autonomous behaviors to utilize fewer tracking mechanisms to drive over road networks. In other words, the path structure encapsulates the important behavioral properties necessary for navigating a route.

Figure 4.15 illustrates three vehicles approaching an intersection. Each vehicle utilizes a path to seamlessly integrate the connection between the lane they are on and the corridor through the intersection.

4.6.1 Path Augmentation and Queries

Paths simplify bookkeeping needed to manage frequent changes in coordinate systems. As intersections are approached, paths grow and shrink by adding on the upcoming corridors and lanes while removing the lanes and corridors at the beginning of the path.

Tracking Queries
$(\delta, o) \Leftarrow$ *queryXY(* $\vec{R^3}$ *)* Converts from Cartesian to curvilinear space
$\vec{R^3} \Leftarrow$ *queryDO(* (δ, o) *)* Converts from curvilinear to Cartesian space.
Feature and Range Attribute Queries
list<RangeAttr> $\Leftarrow$ *queryRangeAttributes(* (δ, o) **)** Returns a list of range attributes along the path
list<Feature> $\Leftarrow$ *queryFeatures(* $(\delta, o), (\delta', o'), \gamma$ *)* Return a list of features along the path including references to stopline and curb corridor attributes
Witness Queries
list<Witness> $\Leftarrow$ *LOD_queryObjects(* δ*, LODProfile)* Returns witnesses occupancy information for the path based on the LODProfile and the given distance parameter (δ)
Witness $\Leftarrow$ *LOD_queryLeader(* δ*, ori, LODProfile)* Returns a witness based on the LODProfile that represents the next object in the direction specified by *ori* after a particular distance, δ, along the path

Table 4.10: Queries provided by paths in the EDF.

Paths can be built incrementally, on an as needed basis by behaviors, using the augmentation routines for appending and removing from the path. Thus, a path is not merely a static description of a route. Although paths can be used to describe specific routes and plans for autonomous behaviors, the intended use of the path is as a local, relative description of the surface of the route. Paths are lane based and not road based. It is up to behaviors to augment and create paths as needed. For instance, during a lane change, the path may have to be reconstructed by the behavior. Table 4.10 lists the queries provided by the path structures in the EDF. Because the path represents a curvilinear surface description, the queries provided by curvilinear surfaces are also supplied by the path, as discussed in Section 4.2.