An Approach to Describe Arbitrary Transition Curves in an IFC Based Alignment Product Data Model

J. Amann[1], M. Flurl[2], J. R. Jubierre[1]and A. Borrmann[1]

[1]Chair of Computational Modeling and Simulation, Faculty of Civil, Geo and Environmental Engineering, Technische Universität München, P.O. Box 80333, Munich, Arcisstrasse 21; PH +49 89 289 23047; FAX +49 89 289 25051; email:{julian.amann, javier.jubierre, andre.borrmann}@tum.de
[2]Chair for Computation in Engineering, Faculty of Civil, Geo and Environmental Engineering, Technische Universität München, P.O. Box 80333, Munich, Arcisstrasse 21; PH +49 89 289 23047; FAX +49 89 289 25051; email: matthias.flurl@tum.de

ABSTRACT

Open standards for infrastructure based on IFC (Industry Foundation Classes) are mainly developed by the openINFRA initiative of the buildingSmart organization. Recently, several proposals for alignment models emerged with the development of the upcoming IFC 5 standard that in particular targets infrastructure projects such as roads, bridges and tunnel buildings. A common drawback of all these proposals is their limited description of arbitrary transition curves. For instance, in all proposed alignment models there are some missing types of transition curves, or different parameters are suggested to describe a certain transition curve type. Designing a neutral data format that satisfies all stakeholders in an international context is therefore difficult. A novel approach to describe transition curves based on the so-called IFCPL (Industry Foundation Classes Programming Language) is described and its integration into an IFC based alignment model is shown to avoid these problems.

INTRODUCTION AND MOTIVATION

The highest level of abstraction in the description of linear infrastructure projects such as roads, bridges or tunnel buildings is the so-called alignment model, which defines the principle course of the infrastructure project. With the ongoing development of the upcoming IFC 5 standard that in particular targets infrastructure projects, many different proposals for alignment models emerged. These proposals share the idea to describe the final alignment by the superposition of two two-dimensional curves, namely the horizontal and vertical alignment. Usually, the horizontal alignment consists of line segments, arcs and transition curves and describes the course of an alignment in the XY plane, while the vertical alignment usually consists of line segments and parabola arcs and defines the corresponding z-coordinates as a function of the length s of the horizontal alignment curve up to a certain point.

Transition curves ensure a smooth transition between elements with different curvature in order to avoid curvature discontinuities (see Figure 1). For instance, if a straight line segment is directly followed by an arc segment, the resulting (continuous) alignment curve would have a discontinuity in the curvature. A possible transition curve is a clothoid also known as Euler spiral. In its real world application a clothoid enables a car driver to ride smoothly through a curve by turning the steering wheel with a constant speed. Besides the clothoid, many different types of transition curves exist that are crucial for smooth alignment design resulting in the difficult question which transition curves should be included in an alignment model.

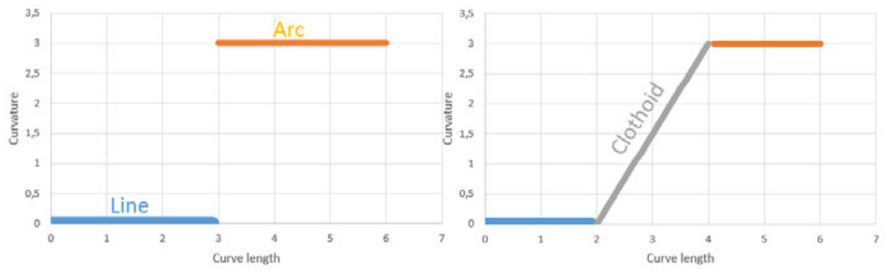

Figure 1. Transitions curves, such as clothoids result in smooth transition from a line to an arc segment which is important in alignment design.

None of the existing standards, such as LandXML (Rebolj et al. 2013), OKSTRA (Schultze and Buhmann 2008), or RoadXML (Chaplier et al. 2010), supports all types of transition curves used in Civil Engineering (see Table 1). For instance, LandXML is missing a C-Clothid curve type, OKSTRA is missing a sinusoid curve type, and RoadXML and the IfcAlignment Proposal (Amann et al. 2013) lack the support of a Bloss transition curve type. Even if we provide a very large amount of different transition curve types, it is most likely that we are missing support of one particular transition curve type that is used for instance in a special domain like railways or is specific to a certain country policy. To overcome these issues, we propose an approach to describe transition curves in a generic representation that supports all transition curve types.

Table 1. Support of different transition curve types across different standards.

Curve Type	LandXML	RoadXML	OKSTRA	IfcAlignment Proposal
Clothoid	Yes	Yes	Yes	Yes
Bloss	Yes	No	No	No
Sinusoid	Yes	No	No	No
Weiner Bogen	Yes	No	No	No
C-Clothoid	No	No	No	No

RELATED WORK

There are a number of existing standards for representing and exchanging alignment data. The most frequently used one for data exchange purposes is LandXML (Rebolj et al. 2013), which is nevertheless currently not being supported by any standard organization. LandXML supports 16 different transition curve types. A transition curve is called 'Spiral' in LandXML. The XML schema definition of a 'Spiral' is shown in Table 2. The development of LandXML suddenly stopped in

2009. Besides, the LandXML schema has several flaws such as insufficient documentation, syntax errors in the LandXML 1.2 schema, weak point typing, case inconsistencies, or name inconsistencies (Scarponcini 2013).

Table 2. Listing of the XSD definition of the element type ‘Spiral’ in LandXML.

```
<xs:element name="Spiral">
  <xs:complexType>
   <xs:sequence>
     <xs:choice minOccurs="3" maxOccurs="3">
       <xs:element ref="Start"/>
       <xs:element ref="PI"/>
       <xs:element ref="End"/>
     </xs:choice>
<xs:element ref="Feature" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="length" type="xs:double" use="required"/>
   <xs:attribute name="radiusEnd" type="xs:double" use="required"/>
   <xs:attribute name="radiusStart" type="xs:double" use="required"/>
   <xs:attribute name="rot" type="clockwise" use="required"/>
   <xs:attribute name="spiType" type="spiralType" use="required"/>
   <xs:attribute name="chord" type="xs:double"/>
```

```
 <--! Continued: -->
 <xs:attribute name="constant" type="xs:double"/>
   <xs:attribute name="desc" type="xs:string"/>
   <xs:attribute name="dirEnd" type="direction"/>
   <xs:attribute name="dirStart" type="direction"/>
   <xs:attribute name="name" type="xs:string"/>
   <xs:attribute name="theta" type="angle"/>
   <xs:attribute name="totalY" type="xs:double"/>
   <xs:attribute name="totalX" type="xs:double"/>
   <xs:attribute name="staStart" type="xs:double"/>
   <xs:attribute name="state" type="stateType"/>
   <xs:attribute name="tanLong" type="xs:double"/>
   <xs:attribute name="tanShort" type="xs:double"/>
    <xs:attribute name="oID" type="xs:string"/>
  </xs:complexType>
</xs:element>
```

The possible 16 different transition curve types are bloss, clothoid, cosine, cubic, sinusoid, reverse biquadratic, reverse bloss, reverse cosine, reverse sinusoid, sine half wave, biquadratic parabola, cubic parabola, Japanese cubic, radioid, Wiener Bogen and can be described by this ‘Spiral’ type (the curve type is defined by the ‘spiType’ attribute). The LandXML ‘Spiral’ types have many drawbacks. For example a clothoid can be described unambiguously by specifying only six parameters (six double values), while LandXML allows to overdetermine the spiral type, for instance by specifying a clothoid with more than six values. Unfortunately, that also allows to specify impossible transition curves, which violates the principle of data integrity.

RoadXML (Chaplier et al. 2010) is a standard that is commonly used in car driving simulation applications. Its road description is also based on a vertical and a horizontal alignment. Exclusively, clothoids are used for transition curves in RoadXML. Clothoids are described by six parameters (see Table 3). This allows a very compact and at the same time unique definition of a clothoid. Other transition curve types are missing in RoadXML.

Table 3. Clothoid parameters in RoadXML.

Name	Type	Description
x	double	The x coordinate of the starting point
y	double	The y coordinate of the starting point
direction	double	The orientation at the starting point
startCurvature	double	The starting curvature (1/radius)
endCurvature	double	The ending curvature (1/radius)
length	double	Length of the clothoid

OKSTRA (Schultze and Buhmann 2008) is used for road information systems for the federal government, states, counties and communities within the Federal

Republic of Germany. OKSTRA does only support clothoids as a transition curve type.

Besides the existing alignment data models, there are also some developments in upcoming standards which also provide their own alignment models like IfcBridge (Yabuki et al. 2006) or openBrIM (U.S. Department of Federal Transportation 2013).

FROM PLAIN PARAMETERS TO CONSTRUCTION RULES

In the following sections we focus on transition curves, but denote them just as curves for simplicity reasons. All subsequent considerations can be easily transferred to curves, in general. All considered standards share the idea to describe a curve by defining a set of parameters, such as a start and an end point, start radius or similar parameters. Thus, if an alignment expert or a software vendor want to leverage the corresponding alignment data model, it is thier duty to understand and interpret these parameters for each curve type. This approach often leads to problems resulting from missing, informal, incomplete or flawed documentation. For example, observe the deficient documentation of the spiral type in LandXML 1.2 schema. The spiral type in LandXML has 19 different XML attributes (see Table 2). For none of them documentation is available (see LandXML documentation at landxml.org). A developer has to deduce the meaning of the attribute merely from its name. But even a very good and complete documentation does not prevent the reader to misinterpret or misunderstand the meaning of given parameters. Besides this problem it is also time consuming to understand and implement each curve type.

A neutral standard for an alignment data model should support a rich set of curve types in order to be accepted internationally, in particular it should be aware of region specific facts. This is a typical problem of all of the described alignment model standards. But even if a rich set of curves is included, it is often not clear which parameters should be used in order to describe a certain curve type. For instance, instead of storing the start or end curvature of a clothoid, the start or end radius could also be used respectively. This burdens the developer of a standard to incorporate a minimal, but sufficient set of parameters or to integrate redundant data, as exemplarily done in the LandXML specification of the spiral type.

Due to the reasons described above, we propose an inversion of control in the design of an alignment model standard. The main idea in the inversion of control approach is not to store parameters and their values solely, but to additionally exchange functions to interpret these values in order to visualize or analyze curves. Thereby, the computational algorithm for a curve itself is described in a neutral data standard – the below proposed so-called IFCPL language – and interchanged between different applications.

INVERSION OF CONTROL – INCORPORATING CONSTRUCTION RULES

In the following section, we will explain this inversion of control approach in detail and highlight its advantages in comparison to the currently used approaches. It is clear, that in order to exchange the geometry of a curve specific parameters and their values must be exchanged. To this end, currently used alignment data models define a proper description for each type of curve and a specific set of parameters, which allows a correct reconstruction of this curve.

To overcome the above explained drawbacks of this approach, we propose to include only one generic curve type into the alignment data model specification representing all possible curves such as clothoids, cubic parabolas, etc. Since different curve types depend on different parameter sets, the way of describing parameters also has to be generic. We propose to use a generic parameter set, which does not dictate which specific parameters to include, but merely gives the possibility to store a set of (suitable) parameters, in a general way. In the field of Building Information Modeling the functionality of *IfcPropertySets* to store parameters and their values, respectively, is widely accepted and standardized. Simply put, an *IfcPropertySet* is a set containing *IfcProperties*, which themselves contain the actual data as a triple including name, type, and value. Thus, we suggest storing the parameter and their values necessary to describe a curve as *IfcProperties* and to compose them into an *IfcPropertySet*.

To allow a correct interpretation of a curve, we determine a mandatory inclusion of suitable functionality in a specific alignment describing document. To this end, a common interface that defines a set of functions necessary to visualize and analyze curves is defined in the alignment data model specification. Furthermore, the specification stipulates the integration of functions implementing the prescribed interface. It remains the user's decision to provide its own suitable interpretation functionality or to rely on already existing functionality. To provide a generic way to create this functionality, we recommend the usage of the so-called IFC Programming Language (IFCPL), which is a dedicated imperative language and will be described below. In particular, this programming language defines a proper interface that is suited to visualize a curve.

This approach offers several advantages:

- The user is free to use whatever set of parameters he or she wants to use to describe a transition curve, as long as a proper interpretation function is available.
- There is no more possibility for the misinterpretation of a given parameter set by the user, since the provided interpretation functionality unburdens the user from the interpretation task.
- A user can introduce new curve types nobody thought of before. Only new functionality to interpret this curve and its describing parameters respectively needs to be added to support this new curve type.
- In principle, no more documentation for describing a curve or the used parameter set respectively is necessary, but surely useful.
- No time is needed to understand and implement different curve types, since this knowledge is no longer vendor specific application knowledge.

For instance, a user describes a clothoid by an *IfcPropertySet* comprising the following parameters: the coordinates of the start point, the tangents intersection point, start and end radius, the clothoid constant and the (clockwise) orientation.

Additionally, he consigns a function that provides the possibility to calculate the x- and y-position of the clothoid points as function of the percentage position between start and endpoint, a functionality to extract the parameters and their values respectively from the *IfcPropertySet*. On the one hand, this enables the recipient to visualize the clothoid by using the first function in an extremely easy, but safe way, and on the other hand, to do its own analysis of the clothoids by providing the specific parameters and their values.

DEFINITION OF AN ARBITRARY TRANSITON CURVE

An interface for a transition curve is shown in Figure 2. This interface is sufficient to display the data of an arbitrary transition curve. The *getLength()* method can be used to query the length of a transition curve segment. The method *getPosition()* is used to retrieve a 2D position on the transition curve (since the transition curve is part of the horizontal alignment, it has a 2D position). When the *lerpParamter* of the *getPosition()* is set to 0 it returns the start position of the transition curve. If it is set to 1, it returns the end position of the transition curve. For values in between **l**inear int**erp**olation (short lerp) is used to determine a point that is located at the corresponding position. For instance, a lerp parameter of 0.3 returns the point that is reached when 30% of the length along the transition curve is travelled. The *getParameter()* method returns the *IfcPropertySet*, which contains all *IfcProperties*.

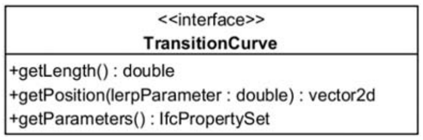
<<interface>>
TransitionCurve
+getLength() : double
+getPosition(lerpParameter : double) : vector2d
+getParameters() : IfcPropertySet

Figure 2. Abstraction of an arbitrary transition curve.

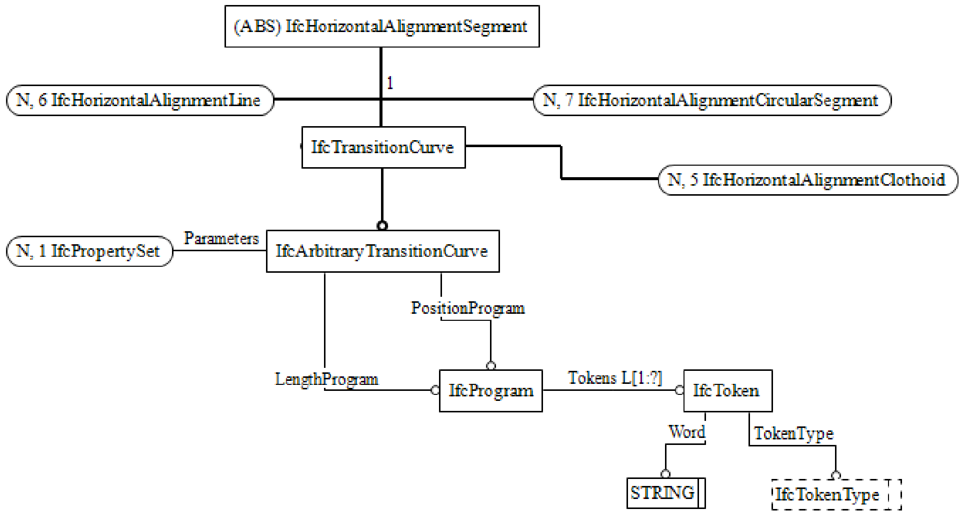
(ABS) IfcHorizontalAlignmentSegment
1
N, 6 IfcHorizontalAlignmentLine
N, 7 IfcHorizontalAlignmentCircularSegment
IfcTransitionCurve
N, 5 IfcHorizontalAlignmentClothoid
N, 1 IfcPropertySet
Parameters
IfcArbitraryTransitionCurve
PositionProgram
LengthProgram
IfcProgram
Tokens L[1:?]
IfcToken
Word
TokenType
STRING
IfcTokenType

Figure 3. Overview of the integration of arbitrary transitions curves into the IFC alignment schema.

A parameter of the transition curve is represented as an *IfcProperty*. The different *IfcProperty* elements are collected in an *IfcPropertySet* that is referenced by an *IfcArbitrayTransitionCurve*. The *IfcArbitrayTransitionCurve* is a subclass of

IfcTransitionCurve. *IfcTransitionCurve* is a part of the alignment model described in [ABH2013]. Furthermore, *IfcArbitraryTransitionCurve* references an *IfcProgram* (see Figure 3). We propose to introduce the entity *IfcProgram* for holding imperative code written in a special-purpose programming language, which will be introduced below. An *IfcProgram* holds the construction rule of a transition curve in the form of a token list. The transition curve is associated with two instances of *IfcProgram* (position and length program). The *IfcProgram* consists of a list of *IfcToken* objects. Each token is a single atomic unit of the language, for instance a keyword, identifier or symbol name.

IFC PROGRAMMING LANGUAGE (IFCPL)

We illustrate the use of IFCPL by an example describing how a curve is defined. For simplicity reasons we choose a straight line. The line is defined by its start and end point. For the start point we store an ‘xStart’ and ‘yStart’ *IfcProperty* of type *IfcPositiveLengthMeasure* in the parameter set (*IfcPropertySet*) of the *IfcArbitrayTransitionCurve*. The same is done for the end point (‘xEnd’ and ‘yEnd’). Now we define two IFCPL programs representing the actual construction rule. To this end, we define the ‘LengthProgram’ and the ‘PositionProgram’. The length program determines how the length of the curve is computed. In the example, the length between two points can be computed by using the Euclidian distance. The length program is depicted in Table 4 (left column).

Table 4. The IFCPL length and position program.

Length program	**Position program**
`dx = xEnd - xStart;`	`vx = (1-lerpParamter) * xStart + lerpParamter * xEnd;`
`dy = yEnd - yStart;`	`vy = (1-lerpParamter) * yStart + lerpParamter * yEnd;`
`dx2 = dx * dx;`	`return(vx,vy);`
`dy2 = dy * dy;`	
`return(sqrt(dx2+dy2));`	

The length program uses the variables ‘xStart’, ‘yStart’, ‘xEnd’, ‘yEnd’. These variables and values are known to the program, because all variables contained in the parameter set (*IfcPropertySet* of the *IfcArbitrayTransitionCurve*) are injected into the length and position program of the transition curve on startup. Additionally, a ‘lerpParameter’ variable is injected into the position program, which is shown in Table 4 (right column). A return statement pushes the computed value on a return stack that can be further processed by an application.

IFCPL provides several basic built in functions like *abs*, *sin*, *cos*, *tan*, *print*, *input*, *return*, *factorial*, or *sqrt*. For instane, the *sqrt* function computes the square root of a number. The print function can be used to write output for debugging purposes. Variables in IFCPL need not to be defined. A variable always takes the type of the expression it is assigned.

SOFTWARE INTEGRATION

For executing IFCPL programs an interpreter is needed that processes the IFCPL code. To validate the proposed approach, we have prototypically implemented an IFCPL interpreter in a tool suite called IFCPL Environment. This environment is provided in the form of a shared library so it can be directly used by other applications such as parametric CAD tools. Thus, there is no need that every tool implements an own parser and interpreter. Figure 4 shows an overview of the IFCPL Environment.

CONCLUSION

We presented a general method for representing transition curves in a generic manner. Our approach is based on the employment of the imperative programming language IFPCL for describing the computation of curve points in dependency of a given abscissa. We indicated the integration of an interpreter of this language into existing software solutions, thus providing a very fast and convenient way for achieving interoperability in the infrastructure sector.

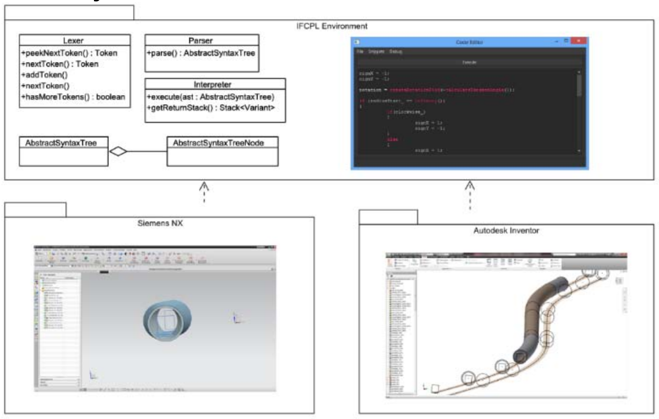

Figure 4. Overview of the IFCPL Environment including the integration of the IFCPL interpreter into Siemens NX and Autodesk Inventor.

REFERENCES

Amann, J.; Borrmann, A,; Hegemann, F.; Jubierre, J.R.; Flurl, M.; Koch, C.; König, M.:(2013). „A Refined Product Model for Shield Tunnels Based on a Generalized Approach for Alignment Representation", In: Proc. of the ICCBEI 2013, Tokyo, Japan.

U.S. Department of Transportation, Federal Highway Administration (2013), "Open BrIM Standards", https://www.transportationresearch.gov/dot/fhwa/ascbt/brim/default.aspx

Chaplier, J., That, T. N., Hewatt, M., and Gallée, G. (2010) "Toward a standard: RoadXML, the road network database format" At the proceedings of the Driving Simulation Conference, pp. 211-220.

Rebolj, D., Tibaut, A., Čuš-Babič, N., Magdič, A., and Podbreznik, P. (2008). "Development and application of a road product model". In: Automation in Construction, Volume 17, pp. 719-728.

Scarponcini, P.: (2013). "InfraGML Proposal (13-121)", OGC Land and Infrastructure DWG/SWG.

Schultze, C. and Buhmann, E. (2008) "Developing the OKSTRA® Standard for the Needs of Landscape Planning in Context of Implementation for Mitigation and Landscape Envelope Planning of Road Projects". International conference on information technologies in landscape architecture; Digital design in landscape architecture, International conference on information technologies in landscape architecture; Digital design in landscape architecture; 310-320.

Yabuki, N., Lebeque, E., Gual, J., Shitani, T. and Li, Z. T., (2006). International Collaboration for Developing the Bridge Product Model IFC-Bridge, In Proc. Of the International Conference on Computing and Decision Making in Civil and Building Engineering.