EG-ICE 2025
GLASGOW

GRC: A Schema for the Analysis of the Complexity of Geometry-based Rule Checks

MARIAN KURZAWA, JAKOB KIRCHNER

University of Kassel, Kassel, Germany

ABSTRACT:
In order for a building to comply with the regulations and building code requirements, there is an increasing necessity for automatic rule checking. A significant proportion of those rules is geometry-based. To date, no generalisation for the implementation or classification of geometry-based rule checks has been established. This paper proposes a schema (GRC) to analyse a geometry-based rule check. GRC is based on taxonomies, which are used to categorise the complexity of the problem and the complexity of the solution. The implementation of the rule check is classified within predefined groups. This enables the evaluation of whether a rule checking implementation is suitable for the given problem or whether a simpler implementation is reasonable and sufficient. The schema also facilitates the comparison of differing rule checking implementations.

KEYWORDS
Rule Check Complexity, Geometry, Automatic Code Compliance Checking

1. INTRODUCTION

The construction and operation of a building require that the building complies with all relevant codes of the various disciplines involved. As construction projects become more complex and the requirements of construction projects increase, automatic code compliance checking becomes more important, particularly in terms of efficiency. Compliance checks usually consist of several semantic or geometry-based checks. Some geometry-based calculations produce numbers, such as the volume of the geometry. However, other geometry-based checks require the computation of auxiliary geometry. In the context of this paper, a geometry-based rule check is defined as a rule check, that requires at least the creation of some kind of auxiliary geometry. Depending on the complexity of the auxiliary geometry, the geometry-based rule checks can differ within a wide range of complexity. In the following, the complexity class of this solution is referred to as the S-Class. It is assigned depending on the complexity of the used geometry commands (see section 4).

This paper proposes a first schema GRC (Geometry-based Rule Classification) for analysing the complexity of geometry-based rule checks. The schema should support a conclusion as to whether a rule check fits the considered test cases. Even a valid implementation of a rule check may be considered too complex, as there are simpler solutions to the problem. On the other hand, it is crucial that the implementation supports the required computations without producing incorrect

Kurzawa, M., & Kirchner, J. GRC: a schema for the analysis of the complexity of geometry-based rule checks. In A. Moreno-Rangel, & B. Kumar (Eds.), EG-ICE 2025: AI-Driven Collaboration for Sustainable and Resilient Built Environments Conference Proceedings University of Strathclyde Publishing. https://doi.org/10.17868/strath.00093250

results due to defect geometry. Therefore, the complexity of the problem (P-Class) and the limitations of the used auxiliary geometry are relevant for the implementation of geometry-based rule checking. The P-Class is determined based on the factors to be considered, such as the level of detail or the dimensions of the geometry (see section 5). The evaluation depends on:

- Which input cases are relevant?
- What can be successfully handled in the implementation?
- What special cases or exceptions need to be considered (e.g., non-manifold geometry)?
- Which of these exceptions lead to incorrect checking results?

A rule check should at least cover all reasonable types of input data. Depending on the input, the requirements on the implementation may vary. If the input only consists of axis-aligned geometry, an implementation does not need to consider non-axis aligned geometry. Therefore, the complexity of the problem (P-Class, see section 5) and the complexity of the solution (S-Class, see section 4) must match.

This paper proposes a schema to facilitate the classification of a geometry-based rule check implementation within certain groups (see sections 4 and 5). Since there are different variants to implement the same rule check, GRC can also be used to compare two implementations. This can help to determine which software product is better suited to the requirements. The simpler implementation can also be identified.

The paper summarises the limitations of an existing classification for rule checking and gives a rough overview of the state of the art in geometry-based rule checking. Then the methodology of this paper is presented. In the following sections, the parts of the schema, the complexity of the solution (based on the commands used in the software) and the complexity of the problem are described. The schema for a classification is explained, using an example from Eurocode 3. The schema is then validated by comparison with other example implementations.

2. STATE OF THE ART OF GEOMETRY-BASED RULE CHECKING

2.1 Limitations of an existing classification for rule checking

A classification for rule checking is given by Solihin, W. & Eastman, C., (2015). In this context, a rule check is considered as a process of checking one or more objects, which results in a logical expression: true usually means, that the object fulfils all given requirements; false means that the object violates one or more requirements. Solihin, W. & Eastman, C., (2015) distinguish four complexity classes for rule checks:

- Rule checks that require explicit data (class 1)
- Rule checks that require simple derived attribute values (class 2)
- Rule checks that require extended data structure (class 3)
- Rule checks that require a “proof of solution” (class 4)

According to this classification, geometry-based rule checks are assigned to class 3 or 4. Since geometry-based rules often require auxiliary geometry, there is a wide range in the complexity of rules within class 3. The rather simple check for sufficient computed room size is class 3. Checking for sufficient clearance space in front of a door is also class 3, although it requires a more complex implementation. The given taxonomy is not sufficient to compare the complexity of geometry-based rule checks. For the new proposed schema, the complexity of the implemented auxiliary geometry is one of the key aspects to analyse the complexity of the rule check itself.

2.2 Geometry-based rule checking

The automation of rule checking is well researched. Several authors discuss the implementation of rule checks, that check explicit data. These use property sets within the Industry Foundation Classes (IFC) to compare them against reference values (Zhang, C. & Beetz, J., 2014 using Model View Definitions). Although some of these use calculations, such as checking the thermal transmittance of windows using the ifcOWL ontology (Beetz, J., *et al.*, 2009) or calculating the sound reduction of walls with ifcOWL (Pauwels, P., *et al.*, 2011), there are fewer examples including auxiliary geometry (class 3 according to Solihin, W. & Eastman, C., 2015). For instance, Jiang, L., *et al.* (2022) develop an ontology-based checking implementation to evaluate the window-to-floor ratio. Depending on the input model, this implementation may require the evaluation of geometric data, such as the window area compared to the floor area.

An example for geometry-based rule checking using auxiliary geometry is defined by Akanmu, A.A., *et al.* (2020). The author uses the game engine Unity to create geometry-based rule checks for maintenance work. The checks include the use of bounding boxes and soft clashes. In this implementation the checks are reduced to finding clashes. The geometry of the intersection itself is not used for further checks. Andrich, W., *et al.* (2022) discuss comparable geometry-based rule checks, such as the check for sufficient clearance

spaces (with Solibri Office, formerly Solibri Model Checker). For these checks, the existence of intersections is sufficient to violate the rule. If the rule check requires the computation of the intersection geometry, it should be considered more complex. Such an implementation is presented by Arias, J., *et al.* (2022). The authors describe an approach for evaluating axis-aligned geometry using constraint logic programming. This approach defines the intersection of two geometries by splitting the geometry into several partial geometries. However, the usability of this implementation is limited by the assumption of axis-aligned geometry.

In addition to rule checks for specific building components, there are also rules that require a more complex combination of multiple concepts. For instance, Stepien, M., *et al.* (2023) introduce an indoor graph network, that requires both the calculation of geometry and the traversal of paths to identify a possible escape route. The complexity of this implementation can vary depending on the requirements for an accessible path, such as a minimum corridor width or sufficient door clearances. To simplify their implementation a generalisation of rule checks is useful. In particular, the geometry-based rule checks mentioned above lack a generalisation for the checking process.

2.3 Formalisation of the rule checking process

A frequently cited generalisation of the checking process is defined by Eastman, C., *et al.* (2009). The process consists of interpreting the given rules, preparing the building model, executing the rules and reporting the results of the check. Another approach to simplify the implementation and execution of rule checks is to formalise the rule expression.

Métral, C., *et al.* (2020) give such a formal description of a rule expression for rule checking with SHACL or SPARQL. According to the authors, the rule for an object within a validity context has a precondition and a postcondition. The conditions have an expression consisting of two operands and an operator (see Figure 1).

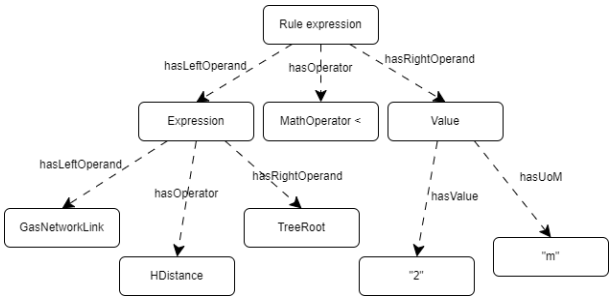

Figure 1: Definition of a rule expression according to Métral, C., et al. (2020).

With this definition, objects can be selected by their validity context and then evaluated by the expression resulting from the output of the operand for the two operators (left and right side). As the concept of Eastman, C., *et al.* (2009) is adaptable for semantic and geometry-based rule checking, the concept of Métral, C., *et al.* (2020) cannot fully cover the commands required for the computation of geometry.

3. METHODOLOGY

In order to derive a schema for the analysis of geometry-based rule checks, key aspects of the complexity of a rule check are abstracted from existing geometry models and geometry languages within software. Two taxonomies are introduced for the analysis of a rule check implementation: the complexity of the problem (P-Class) and the complexity of the solution (S-Class).

The P-Class takes into account the complexity of the input geometry and its underlying model. For example, if a geometry-based problem is only relevant for axis-aligned geometry, the requirements for the implementation are lower than for non-axis-aligned geometry.

The S-Class considers the used commands, as there are different variants to implement a geometry-based rule check. For instance, a command may create non-manifold geometry (e.g., self-intersecting shapes). Depending on the implementation, this defect geometry may produce incorrect checking results.

The analysis schema is based on the two developed taxonomies. The taxonomies only represent the proposed concept and they are not necessarily complete or exhaustive. To compare different implementations, it may be reasonable to add other criteria.

Both the taxonomies (P-Class, S-Class) and the schema GRC are explained and validated with geometry-based example rule checks from building codes. They cover several fields of civil engineering, such as structural engineering, accessibility of buildings, infrastructure and excavation works, to demonstrate the usability of GRC. The example rules are generated twice with different implementations. Both implementations are classified and then compared.

4. COMPLEXITY OF THE SOLUTION (S-CLASS)

As defined by Métral, C., *et al.* (2020), a rule expression has left and right operands that are combined by a mathematical operator. The author uses the model to check simple attribute values. However, this construct can be abstracted to represent any more complex rule expression, as the operands can be rule expressions themselves.

A complex rule expression can be composed by concatenating simpler expressions. These simpler expressions can be divided into three different functional categories: CREATE, SELECT, CALCULATE. Figure 2 shows commands that can be abstracted from OpenSCAD, GDL (Geometric Description Language in ArchiCAD), Dynamo, and Borrmann, A. & Rank, E. (2009). This collection is not complete. It is reduced to the most important commands identified.

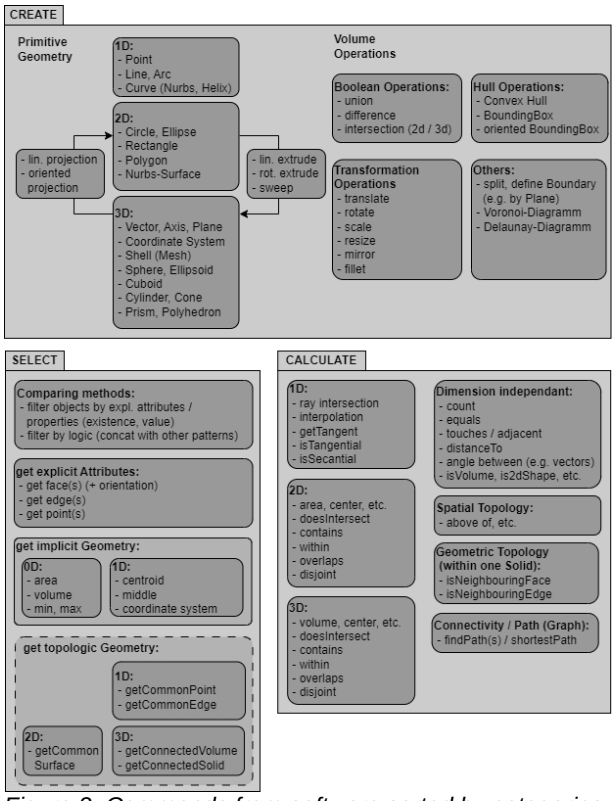

Figure 2: Commands from software sorted by categories.

To classify the complexity of the commands, mainly the CREATE category is relevant, as these commands allow more complex computations than the others. Although there is no natural taxonomy within these commands, there are reasonable assumptions to classify them. For example, the use of sweeps allows more complex computations than the creation of a simple geometric primitive, such as a cube. Boolean difference is also considered to be more complex than the union, as the difference can create solids with collapsing dimensions or disjoint geometries. Depending on the implementation, this can lead to incorrect checking results.

It should also be noted that some of these commands are not comparable in terms of complexity. For example, whether creating a sweep is more complex than checking for touching surfaces or intersections of geometry depends heavily on the used input data and the following commands. In some cases, the computation of a sweep can be considered challenging, while in other cases the interpretation of the spatial topology can lead to incorrect results, due to a not sufficient level of detail. Consequently, the commands are divided into groups (see Table 1). A rule check can have different complexity classes for the S-Class. Each group can be classified by the used command with the highest complexity. A comparison of different rule checks is only meaningful if they are within the same group.

To improve the expressiveness of the classification, the complexity of the problem (e.g., only axis-aligned models) should be considered as a second key aspect.

Table 1: Taxonomy for the solution (S-Class) based on geometry-based commands in CAD and modelling software.

Group	Command	S-Class
Geometry (simple)	Primitives	1
	Polygon	2
	Prism, Polyhedron	3
Geometry (computed)	Transformation (e.g., rotate)	4
	Bounding Box	5
	Projection	6
	Sweep (except Blend, Loft)	7
	Boolean Union	8
	Boolean Difference	9
	Blend, Loft	10
	Hull Operations	11
Surfaces	Mesh	1
	Shell (BREP)	2
Spatial Topology	Equal to, Distance to	1
	Overlaps, Touches, Intersects, Inside / Contains	2

5. COMPLEXITY OF THE PROBLEM TO CHECK (P-CLASS)

The complexity of the problem is not necessarily equivalent to the complexity of the implementation. For example, the height of excavation pits based on the German Building Code (DIN 4124:2012-01) can be checked for a 3D model. While a check with solids requires 3D intersection tests, the problem can also be reduced to a 2D problem (see Figure 3), resulting in a single check of the height coordinates for the points. In addition to the **dimension** of the problem, the **orientation and shape** of the geometry are also relevant.

A collision check for orthogonal, or even axis-aligned, convex geometry has fewer special cases or exceptions to consider than a check for concave or randomly oriented geometry. Curved surfaces are also more complex to handle than meshed or planar surfaces.

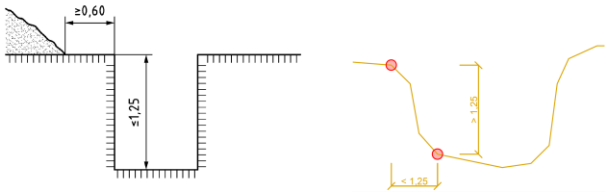

Figure 3: Excavation pits according to DIN 4124 (left) and a simplified height calculation (right).

As mentioned earlier, in some cases geometry-based operations can result in the creation of defective or **non-manifold geometry**. In a mesh, missing surfaces may not be critical. However, when considering boolean operations on solids, watertight, not-self-intersecting geometry is crucial. It depends on the implementation, whether non-manifold geometry causes the implementation to produce incorrect results or whether objects violate the rule to check.

Apart from pure geometric complexity, the general complexity also depends on the **input model**. For example, an indoor navigation problem can be solved with less complexity, if an accessibility graph, or at least relations between rooms and doors are within the given input data model. The navigation problem also shows another influencing aspect. To determine, whether a path is accessible for a wheelchair user, the path must satisfy several requirements, such as the minimum path width and sufficient manoeuvering space in paths and in front of doors (see DIN 18040-1 and 2010 ADA Standards for Accessible Design). A **linear** approach is often not sufficient to solve such problems. Therefore, the computational complexity (see Big O notation) is also relevant to the analysis of the problem complexity.

Analogous to the S-Class, within the P-Class there are different groups for evaluating the complexity of implementations, which should not be mixed. For example, it is not reasonable to compare the computational complexity of a problem with the alignment of the input geometry. As with the S-Class, the comparison for the P-Class is only defined for classes in the same group. A list of these groups and criteria is given in Table 2.

6. SCHEMA FOR COMPARISON

The developed analysis schema GRC is explained and validated with example rule checks from the building codes. The building code requirements are implemented in two different variants and the complexity of the implementations is compared using GRC. The first example is explained in detail to illustrate the use of the schema. The following three examples validate the schema. Their explanations are less detailed. They focus on the differences between the variants.

Table 2: Taxonomy for the problem (P-Class) based on the occurring geometry.

Group	Specification	P-Class
Precision	Integer	1
	Decimal	2
Level of Detail	Planar / Straight	1
	Meshed / Segmented	2
	Curved	3
Alignment	Voxelised / Equidistantly	1
	Axis-Aligned	2
	Orthogonal	3
	Non-orthogonal	4
Dimension (main checks)	1D	1
	2D	2
	3D	3
Shapes	Joint, Convex	1
	Disjoint, Convex	2
	Joint, Not Convex	3
	Disjoint, Not Convex	4
Non-Manifold	Gaps, Clashes	5
	Self-Intersections	5
	Collapsing Dimensions	5
	Not Watertight	5
Semantics	Necessary Relations stored	1
	Necessary Relations missing	2
Computational Complexity	Linear	1
	Quadratic, etc.	2

6.1 Building code example 1: requirements for the placement of reinforcement

To ensure the durability of reinforced concrete structures, the Eurocode 2 requires sufficient cover for the reinforcement (see DIN EN 1992-1-1:2011-01 4.1). The amount of cover required depends on the exposure class of the concrete (risk of corrosion by carbonation, chlorides, etc.), and the minimum cover and bond requirements. Although the amount of cover required varies depending on the material and other factors, the amount itself is not important for the implementation, as the implementation does not become more complex as the reinforcement cover increases. The checks are based on a 35 mm cover (C25/30, XC1). In addition, the spacing of the bars is defined (see DIN EN 1992-1-1:2011-01 8.2). The minimum clear distance between single parallel bars should be sufficient to fill the concrete aggregate. This depends on the diameter of the reinforcement and the maximum size of the aggregate (assumption: a = 2 cm plus diameter). These two checks for the reinforcement position are just examples of such geometry-based rule checks. Similar rules can be formulated for the anchorage length, the design lap length and other geometric requirements.

For the checks, a simple example model is created in Autodesk Revit 2025 (see Figure 4). The model contains three axis-aligned beams and columns. For simplicity, the slab is not modelled.

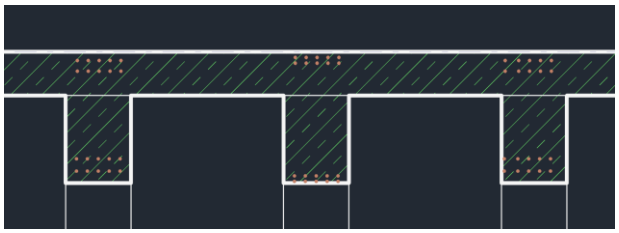

Figure 4: Revit example with sufficient (on the left) and insufficient cover and clear distance between the bars.

In Revit, the rebars are stored as lines that define the basic shape of the reinforcement. The placement of the reinforcement is different for each beam. The implementations are created in Dynamo v3.3.3.0 (see Figure 5). The reinforcement checks are performed using the "Structural Design" package.

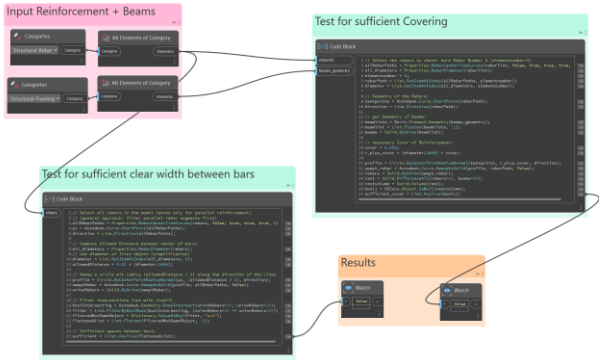

Figure 5: Code for implementation Bar_A in Dynamo.

The concept of implementation Bar_A:

In the first implementation, the reinforcement coverage is checked using 3D auxiliary geometry. For each rebar (line geometry) a cylindrical auxiliary volume is created with the radius of the required coverage plus the radius of the rebar (sweep circle along path). For sufficient coverage, the created shape must be completely inside the beams (see Figure 6). Alternatively, the boolean difference between the created auxiliary cylinder and the beam volumes should be zero. The check for sufficient clear spaces between the bars can be implemented similarly. The lines of the bars are swept with the radius of the required clear distance plus the radius of the bars. The distance meets the requirements, if there is no intersection between the auxiliary geometries (see Figure 6). As each geometry is checked against each other, a check against itself should not be considered.

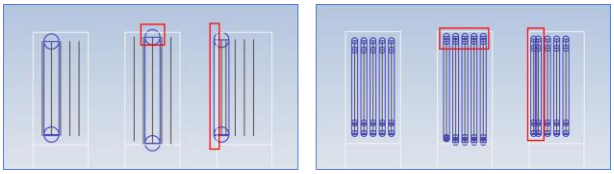

Figure 6: Volumetric concept for implementation Bar_A (left: sufficient covering; right: sufficient clear distance between bars).

The concept of implementation Bar_B:

The second implementation is based on 1D checks. As the geometry of the bars is modelled as parallel lines, a 1D approach can be sufficient for the checks. To check for sufficient coverage, each point of the lines must have the required minimum distance to the boundaries of the beam. As this example deals with linear reinforcement parallel to the surfaces of the beam, it is sufficient, to check one point on each line. For each starting point of a line, a new line with the length of the required coverage plus the radius of the bar is created orthogonally. For sufficient coverage, none of these lines should intersect any of the boundary surfaces of the beam (see Figure 7). The second check includes the distance between the lines representing the rebars.

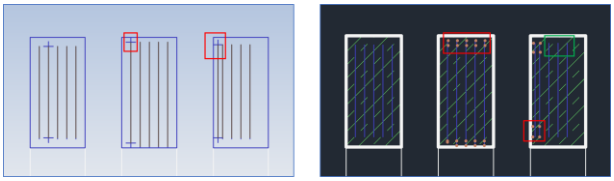

Figure 7: Line-concept for implementation Bar_B (left: sufficient covering; right: sufficient clear distance between bars).

Comparing both implementations:

The two implementations can be classified using the two proposed taxonomies (see Tables 1 and 2). Based on the classifications, the implementations are compared and evaluated using the highest complexity class of the group. The comparison for the reinforcement example is shown in Table 3. Only the groups with different complexity are relevant for the comparison. For instance, precision, semantics and level of detail in the input model are not considered relevant, as they effect both implementations and can be handled by both of them. The comparison highlights the difference in complexity in terms of the dimensions of the used auxiliary geometry (3D, 1D). The geometry operations are significantly more complex for implementation Bar_A (class 9, class 1). Implementation Bar_A also deals with collapsing dimensions, as with boolean difference a resulting volume of zero is possible. In this case, the collapsing volume is intended and does not

lead to incorrect checking results. In addition to these drawbacks in terms of S-Class complexity, implementation Bar_A can deal with non-orthogonal shapes, whereas implementation Bar_B requires reinforcements aligned parallel to the surfaces of the beam. This example shows that such a comparison with GRC is possible, but does not necessarily provide an absolute statement as to which implementation is preferable. Depending on the input model implementation, Bar_A and Bar_B may be more suitable than the other.

Table 3: Comparison of the two implementations for building code example 1 – the placement of reinforcement.

Group	Bar_A	Class	Bar_B	Class
Geo.	Primitives (Circle)	1	Primitives (Line)	1
	Sweep	6		
	Boolean Union	7		
	Boolean Difference	**9**		
Spatial Top.			Distance	1
			Intersects	**2**
Align.	Non-orthogonal	**4**	Axis-Aligned (beam || bar)	2
Dim.	3D	**3**	1D	1
Shape	Collapsing Dimensions	**5**	Joint, Convex	1

6.2 Building code example 2: requirements for the accessibility of doors

Several national building codes define accessibility requirements for public buildings. In this example, the focus is on the accessibility of doors. The German Building Code DIN 18040-1 and the corresponding American Building Code 2010 ADA Standards for Accessible Design require door clearances in front of doors (usually in a size of 1,5 x 1,5 m) for wheelchair use. For simplicity, the following implementations assume, that the clearances are centered in front of the doors. Sufficient clear space is demonstrated by checking that the clear spaces do not interfere with other building components, in this case columns.

Two variants to create the clearance spaces:

There are two implementations for this rule check. While the main concept is the same in both implementations, the auxiliary volumes for the clearances are created differently.

In the first implementation (see Figure 8), a bounding box of the door is created. This auxiliary bounding box is then scaled to 1.5 x 1.5 m (scale factor: 1.5 / door with; 1.5 / wall thickness). One copy of the box is translated in the opening direction of the door and one copy is translated in reverse direction. Both represent the door clearances. They are checked for collisions with all columns and for each collision, the causing door is filtered.

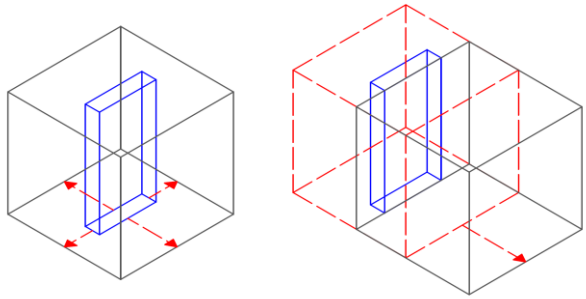

Figure 8: Implementation Door_A - the door clearance is created by translating a scaled bounding box of the door.

The second implementation calculates the coordinates of the point in front of the door. A copy of this point is translated by 1.5 m in the opening direction of the door. Copies of both points are translated in the z direction by the door height (see Figure 9). A rectangular surface is created from these points, which is then thickened by 1.5 m to form a solid. The resulting clearance is mirrored on the other side of the door. The collision check is implemented analogously to the first implementation.

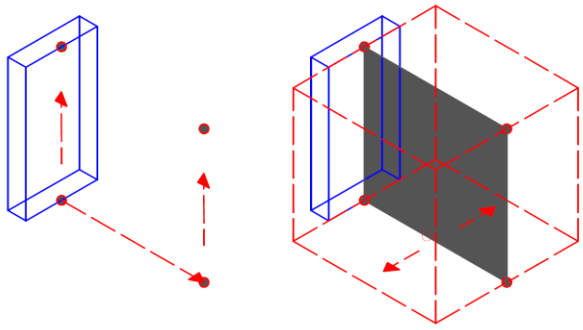

Figure 9: Implementation Door_B - the door clearance is created by thickening a surface made of 4 points.

Comparing both implementations:

The two implementations are classified, using the proposed schema. The result is shown in Table 4. They are considered equal in most of the defined groups, since they require the same geometric and semantic input data. There is no difference in terms of planarity, orthogonality or convexity. Both implementations require 3D intersection checks. However, they differ in the creation of the auxiliary geometry. While the first implementation uses a scaled bounding box, the second creates a rectangular surface and thickens it to create the

solid. Theoretically, the second implementation would lead to self-intersecting and therefore non-manifold geometry, if the points of the surface were sorted in the wrong order. As these points are always created in the correct order and manually aligned for the surface creation, this issue is not considered problematic. As there is no difference in complexity for the most complex geometry-based operation, the two implementations are considered equal in terms of complexity.

Table 4: Comparison of the two implementations for building code example 2 – the accessibility of a door.

Group	Door_A	Class	Door_B	Class
Geo.	Transform. Scale	4	Primitives (Rectangle)	1
	Bounding Box	5	Sweep (Thicken)	7
	Boolean Union	**8**	Boolean Union	**8**

6.3 Building code example 3: requirements for the visibility of railway signals

The third example comes from the field of infrastructure. Sufficient visibility of railway signals on railway lines, is essential to minimise the probability of train crashes or derailments. Depending on the speed of the train, railway signals should be visible from a distance of 300 to 500 m (Holzfeind, J., *et al.*, 2025; Table 12.5).

A simplified example is implemented, where a single-track railway is defined by a polycurve (multiple lines). The model also contains the geometry of a railway signal. The simplified rule implementation should check, whether the railway signal is visible from any point on the railway line that is less than 500 m ahead of the railway signal. To take into account the position of the train driver, the height of the test point is increased by 2 m in z-direction. A 2 x 1 m rectangular area is created for the windshield to represent the field of vision.

Two variants to create the field of vision:

In the first implementation (see Figure 10), the point of the train driver and the position of the signal are connected to form a line that is extruded by the created surface. Such a sweep is created for each starting point of the train line segments. It is assumed, that the length of the segments is small enough to give a correct checking result. Otherwise, more test points on the line segment have to be checked. It is also mandatory, to filter those test points, that have a distance to the signal of less than 500 m. It should also be checked that the train line is facing towards the signal and vice versa. The created auxiliary sweeps are checked for collisions with other geometry objects, in this case walls.

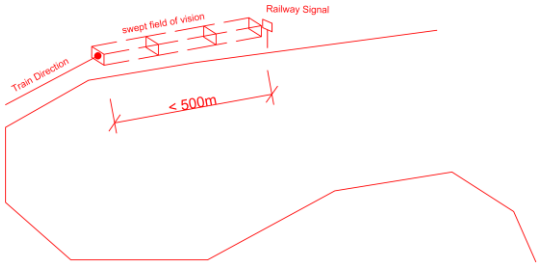

Figure 10: Implementation Sign_A - the field of vision is created by a swept rectangle.

A second implementation is implemented without the use of extruded solids, but with lofted auxiliary solids (see Figure 11). This geometry-based operation can be considered more general, as the generated geometry can have different cross sections at the start and end of the sweep.

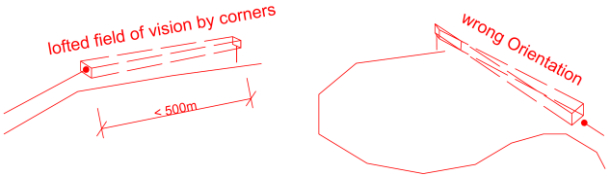

Figure 11: Implementation Sign_B - the field of vision is created by a lofted volume. If the railway signal is oriented into the wrong direction, the auxiliary volume is not watertight.

Comparing both implementations:

Both implementations have a higher complexity than the implementations for building code example 2, as there are a number of test points that need to be checked against a number of geometries. The computational complexity is assumed to be quadratic.

However, similar to building code example 2, the lofting requires the correct order of lines for the cross sections. The order could be incorrect if the signal is not oriented towards the test point. The resulting geometry would not be watertight. Also, a valid geometry is not necessarily planar, but can have curved surfaces. As Table 5 shows, implementation Sign_B is considered to be more complex than implementation Sign_A.

Table 5: Comparison of the two implementations for building code example 3 – the visibility of railway signals.

Group	Sign_A	Class	Sign_B	Class
Geo.	Primitives (Line, Rect.)	1	Primitives (Line, Rect.)	1
	Extrusion	7	Loft	**10**
Detail	Planar	1	Curved	**3**
Shape	Joint, Convex	4	Not watertight	**5**

6.4 Building code example 4: excavation pits

The fourth building code example checks the safety of excavation pits according to the German Building Code (DIN 4124:2012-01, see Figure 3). The input for the rule check is a topography consisting of points. For the checks, it is assumed, that the topography has enough points at the excavation pits (distance between points less than 1.25 m). If this is not the case, additional points can be calculated by interpolating the existing points. To simplify the checks, a slope of more than 45 degrees between two points within an xy distance of 1.25 m is considered to violate the rule (see Figure 12).

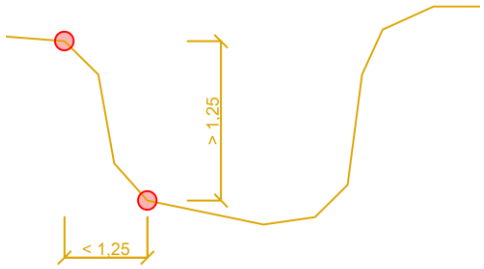

Figure 12: Implementation Pit_A - calculation of the slope and height difference using points.

Two variants to check the height difference:

Implementation Pit_A checks whether two points have an xy distance lower than 1.25 m and a height difference greater than 1.25 m. If the heigh difference is greater than the xy distance, the slope is greater than 45 degrees and the lower point is considered to violate the rule check.

Another implementation can be achieved by creating an auxiliary cylinder 1.25 m above the test point (see Figure 13). If any other topographic point is positioned inside the cylinder, the point violates the rule.

Table 6: Comparison of the two implementations for building code example 4 – excavation pits.

Group	Pit_A	Class	Pit_B	Class
Geometry	Primitives (Point)	**1**	Primitives (Cylinder)	**1**
Spatial Top.	Distance to	**1**	Contains	**2**
Dimension	1D	**1**	3D	**3**

Comparing both implementations:

Both implementations use topography meshes that are not watertight. As the checks themselves are based on points rather than solids, the non-watertight geometry of the topography is not critical. The checks are performed for each point with every other point. The computational complexity is quadratic. However, the implementations differ in the dimensions of the used auxiliary geometry. The first implementation uses only points and distance calculation for 1D geometry, while the second one uses auxiliary volumes (cylinders). The second one also has to check, whether a point is inside of a cylinder, which would require 3D calculations. The result of the comparison is shown in Table 6.

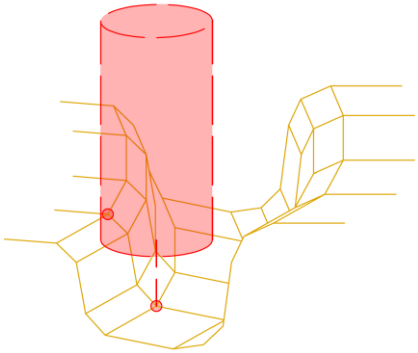

Figure 13: Implementation Pit_B – rule check using cylindrical auxiliary volumes (check: point in cylinder).

6.5 Results of comparing the implementations

The examples show that a comparison between implementations is only reasonable within the same group. This is the case for comparisons between two implementations of the same rule check. These differ in particular in the complexity of the commands for creating geometry. As both implementations handle the same problem, typically there is no difference between the computational complexities of the compared implementations.

However, when comparing implementations of different rule checks, the computational complexity may be relevant. For example, the computational complexity of building code example 4 exceeds the computational complexity of building code example 2 because in example 2 only one door needs to be checked against all columns (linear), whereas in example 4 every point needs to be checked against every other point (quadratic). Also, only the relevant groups need to be considered. For building code example 4, the not-watertight geometry is irrelevant. For building code example 2, not-watertight volumes can lead to incorrect rule checking results.

7. DISCUSSION AND OUTLOOK

The discussed building code examples show that GRC facilitates the classification of geometry-based rule checks. The classification indicates the limitations of a given implementation. The schema can assess, whether the implementation of the rule check is suitable for the given problem. This also helps to assess, whether the implementation is

considered efficient or whether a simpler implementation is preferable, for instance using 1D point distances instead of 3D volume operations (see building code example 4).

GRC also supports the comparison and differentiation of implementations for the same rule check. The comparison does not lead to a natural rating, as an implementation may be more complex for one group, but simpler for the other. The question of which implementation is more appropriate depends on the context. The evaluation may reveal differences in terms of the complexity of the geometry-based operations as well as possible conflicts with defect geometry. If two implementations produce the same results, but differ in terms of complexity, the less complex implementation can be identified for a particular group. Of course, the comparison requires sufficient taxonomies for the P-Class and S-Class.

The proposed schema can be extended to other groups and it highlights possible problems with geometry-based implementations. Therefore, GRC supports the implementation of geometry-based rule checks and the selection of fitting implementations for a given problem. The schema focuses on geometry-based rule checking, but it can also be used for general evaluation of geometry-based algorithms.

REFERENCES

Akanmu, A.A., Olayiwola, J. & Olatunji, O.A., (2020). Automated checking of building component accessibility for maintenance. *Automation in Construction*, 114: p. 103196. DOI: 10.1016/j.autcon.2020.103196.

Andrich, W., Daniotti, B., Pavan, A. & Mirarchi, C., (2022). Check and Validation of Building Information Models in Detailed Design Phase: A Check Flow to Pave the Way for BIM Based Renovation and Construction Processes. *Buildings 2022*, 12(2). DOI: 10.3390/buildings12020154.

Arias, J., Törma, S., Carro, M. & Gupta, G., (2022). Building Information Modeling Using Constraint Logic Programming. *Theory and Practice of Logic Programming*. 22(5): p. 723-738. DOI: 10.1017/S1471068422000138.

Beetz, J., van Leeuwen, J. & Vries, B.d., (2009). IfcOWL: A case of transforming EXPRESS schemas into ontologies. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* 23(01): p. 89–101. https://doi.org/10.1017/S0890060409000122.

Borrmann, A. & Rank, E., (2009). Specification and implementation of directional operators in a 3D spatial query language for building information models. *Advanced Engineering Informatics*, 23(1): p. 32–44. DOI: 10.1016/j.aei.2008.06.005.

DIN Deutsches Institut für Normung e.V., (2011). DIN EN 1992-1-1: Eurocode 2: Bemessung und Konstruktion von Stahlbeton- und Spannbetontragwerken – Teil 1-1: Allgemeine Bemessungsregeln und Regeln für den Hochbau; Deutsche Fassung EN 1992-1-1:2004 + AC:2010. *Berlin: Beuth Verlag*.

DIN Deutsches Institut für Normung e.V., (2012). DIN 4124: Baugruben und Gräben – Böschungen, Verbau, Arbeitsraumbreiten. *Berlin: Beuth Verlag*.

DIN Deutsches Institut für Normung e.V., (2010). DIN 18040-1: Barrierefreies Bauen – Planungsgrundlagen - Teil 1: Öffentlich zugängliche Gebäude. *Berlin: Beuth Verlag*.

Eastman, C., Lee, J.-m., Jeong, Y.-s. & J.-k. Lee, (2009). Automatic rule-based checking of building Designs. *Automation in Construction*, 18(8): pp. 1011–1033. DOI: https://doi.org/10.1016/j.autcon.2009.07.002.

Holzfeind, J., Liu, J., Pospischil, F., (2025). Handbuch Eisenbahninfrastruktur, 4. Auflage. *Springer Viewig*. DOI: https://doi.org/10.1007/978-3-662-68960-8

Jiang, L., Shi, J. & Wang, C., (2022). Multi-ontology fusion and rule development to facilitate automated code compliance checking using BIM and rule-based reasoning. *Advanced Engineering Informatics*, 51(2): 101449. DOI: 10.1016/j.aei.2021.101449.

Métral, C., Daponte, V., Caselli, A., Di Marzo Serugendo, G. & Falquet, G., (2020). ONTOLOGY-BASED RULE COMPLIANCE CHECKING FOR SUBSURFACE OBJECTS, *ISPRS – Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci.*, vol. XLIV-4/W1-2020, pp. 91–94. DOI: 10.5194/isprs-archives-XLIV-4-W1-2020-91-2020.

Pauwels, P., Deursen, D.V., Verstraeten, R., De Roo, J., De Meyer, R., Van de Walle, R. & Van Campenhout, J., (2011). A semantic rule checking environment for building performance checking, *Automation in Construction*, 20(5): pp. 506–518. DOI: https://doi.org/10.1016/j.autcon.2010.11.017.

Solihin W. & Eastman, C., (2015). Classification of rules for automated BIM rule checking development. *Automation in Construction*, 53: pp. 69–82. DOI: 10.1016/j.autcon.2015.03.003.

Stepien, M., Vonthron, A. & König, M. (2023). OpenBIMRL – An open format for code compliance checking of complex functional requirements from a regulation to the BIM model. 30th EG-ICE: International Conference on Intelligent Computing in Engineering at London.

U.S. Department of Justice Civil Rights Division Disability Rights Section, (2010). 2010 ADA Standards for Accessible Design. *United States of America*.

Zhang, C. & Beetz, J., (2014). Model View Checking: Automated Validation for IFC Building Models. *ECPPM 2014 at Vienna*. https://doi.org/10.1201/b17396-24.