4.2 Results of the NoSQL Geospatial Criteria Comparison Matrix

Table 2 exhibits the results of the selected NoSQL DBMSs against the researcher created weighted criteria. It is evident by the scores that MongoDB, Elasticsearch and Solr have the most suitable spatial data management functionalities for the requirement of the study. As a result, they will be reviewed in the next section. Every database apart from the HBase (not HBase Spatial) supports the storage of at least point geometries and only Neo4J supported 3D and only points. In the case of Cassandra and Neo4j when a spatial functionality is not natively supported there is a possibility of adding external libraries to extend spatial functionalities.

Table 2 NoSQL Geospatial Criteria Comparison Matrix. Data sourced from: MongoDB, 2021; Amazon DynamoDB, 2021; Amazon DynamoDB, 2021; Cassandra, 2021; Elasticsearch , 2021; HBase , 2021; Neo4j , 2021; Solr , 2021; Splunk, 2021; Redis, 2021

NoSQL Geospatial Criteria Comparison Matrix												
	Criterion	Weighting	1. MongoDB	2. Redis	3. Elasticsearch	4.Cassandra	5.Splunk	6. Amazon DynamoDB	7.Neo4j	8.Solr	9.HBase	10.Microsoft Azure Cosmos DB
1	Supports point geometry	3	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓
2	Supports linestring geometry	3	✓	✗	✓	✓	✓	✗	✓	✓	✗	✓
3	Supports polygons geometry	3	✓	✗	✓	✓	✓	✗	✓	✓	✗	✓
4	Supports topological queries	2	✓	✗	✓	✓	✗	✗	✗	✓	✗	✓
5	Supports storage of JSON and GeoJSON	2	✓	✓	✓	✗	✗	✓	✗	✓	✗	✓
6	Documentation on performing spatial queries	2	✓	✓	✓	✓	✓	✗	✓	✓	✗	✓
7	Supports spatial Indexing	1	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓
8	Count of spatial operations		4	5	4	3	1	6	5	2	0	3
9	Supports 3D geometries	1	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗
10	Supports embedding user created functions into the DBMS	3	✓	✓	✓	✓	✓	✗	✓	✓	✗	✗
	Weighted score		19	8	19	15	15	6	16	19	0	16
	Key		Supported ✓ | Not Supported ✗ | Supported using a Library ✓ | Low Importance | Medium Importance | High Importance |									

4.3 State-of-the-art Review of Geospatial Features in NoSQL databases and Selected Relational Database

Table 3 summarises the information of PostGIS, an extension of the open-source RDBMS PostgreSQL and NoSQL platforms Elasticsearch, Solr and MongoDB.

All platforms are compatible with multiple operating systems as well as enabled API support. APIs act as vital intermediaries between geospatial applications and data stored in the DBMSs. However, all NoSQL databases in Table 3 have a unique query language for data management. This dramatically reduces the interoperability between systems. If users wanted to retrieve data from multiple databases, they would have to learn the equivalent query in each language, whereas SQL is essentially universal.

Each DBMS presented in Table 3 supports the basic vector geometries. PostGIS has greater capabilities and supports 3D and 4D (3D with a measurement). The stored 3D can be queried by a suite of 3D spatial operators which can be found in the PostGIS documentation (PostgreSQL, 2021).

Each of the three NoSQL databases in Table 3 uses at minimum a Geohash-based index for geometries. The Geohash index only encodes latitude and longitude pairs, limiting the storage to two-dimensional geometries and unprojected geographic coordinates (Only WGS84 CRS). Another significant constraint of the Geohash index is the arbitrary precision given to values (Krommyda & Kantere, 2020) which can further reduce the accuracy of geospatial calculations. PostGIS, on the other hand, uses multiple indexes for general data but GiST for spatial indexing. GiST indexing enables the DBMS to support thousands of SRIDs in. There is no consensus amongst researchers on which is more optimised for indexing spatial geometries. This is apparent in Section 3.5 when reviewing the results of spatial performance comparisons between MongoDB, which uses Geohash and PostGIS/PostgreSQL, which uses GiST.

Table 3 Comparison of geospatial functionality in DBMSs. Data sourced from: Solid IT, 2021; Solr, 2021, Elasticsearch , 2021; Guo & Onstein, 2020; PostgreSQL, 2021

Database	NoSQL			Relational Database
	MongoDB	Solr	Elasticsearch	PostGIS
Database Model	Document store	Search engine	Search engine	Relational database
Developer	MongoDB, Inc	Apache Software Foundation	Elastic	PostgreSQL Global Development Group
Server Operating System (OS)	Linux, OS X, Solaris and Windows	All OS with a Java VM	All OS with a Java VM	FreeBSD, HP-UX Linux, NetBSD, OpenBSD OS X, Solaris, Unix and Windows
Query Language	MongoDB Query Language (MQL)	Lucene query syntax	Domain Specific Language (DSL)	SQL
APIs	RESTful HTTP/JSON API	Java API, RESTful HTTP/JSON API	Java API RESTful HTTP/JSON API	ADO.NET ,JDBC, native C library ODBC, streaming API for large objects
Supported Geometry Types	Point, linestring, polygon, multipoint, multilinestring, multipolygon, geometrycollection	Points, circles, envelopes, line strings and polygons	Point, linestring, polygon, multipoint, multilinestring, multipolygon, geometrycollection	Point, linestring, polygon, multipoint, multilinestring, multipolygon, geometrycollection, 3DM,3DZ and 4D
Spatial Operators	$geoIntersects, $geoWithin, $near, $nearSphere	geofilt, bbox	geo_shape, geo_bounding_box, geo_distance and geo_polygon	Over 1000, see PostgreSQL (2021) for full list
Coordinate Type	Geographic coordinates, only WGS84	Geographic coordinates	Geographic coordinates	Over 3000 SRID
Spatial Indexes	2dsphere index, 2d index-based on Geohash	Prefix tree, Geohash, and quadtree	GeohashPrefixTree and QuadPrefixTree	GiST and SP-GiST
FME Compatible	Yes	No	Yes	Yes

4.4 Final NoSQL Selection

To ensure that a spatial performance comparison between a NoSQL database and RDBMS is extensive, the spatial operators in queries must be diverse and support a range of geometries. For example, in Table 3, Solr's spatial operations is limited to only a distance filter and a bounding box operation. Similarly, the spatial operators in Elasticsearch, apart from geo_shape only retrieve point geometries that intersect a shape. In contrast, all MongoDB spatial operators return all the supported geometry types. Therefore, the influence of geometry complexity in query response times can be measured; This characteristic is the prime reason why MongoDB was chosen for performance comparison.

4.5 NoSQL Selection Summary

In a review of the qualitative analysis, Table 2 shows that amongst the most widely used NoSQL databases, there is a consensus that users need to store spatial data. However, its management in the form of spatial operators is still in its infancy. Table 3 exemplifies the inadequacy. The RDBMS has over 1000 spatial operators, whereas the NoSQL DBMSs only had between 2-4. As a result of the selection process, the NoSQL DBMS MongoDB will be used in spatial performance comparison in Sections 5 and 6.

5 Methodologies of the Experiments

This Section presents the methodology of the three spatial performance comparisons experiments between MongoDB and PostGIS/PostgreSQL. Figure 6 displays how the experiments link to the software selected in Section 4. To present a concise account of the research, at least one example of each Experiment will be presented to demonstrate how the methodology was executed, further detail is available in the appendix. A number of results will be presented in this Section to clarify subsequent steps.

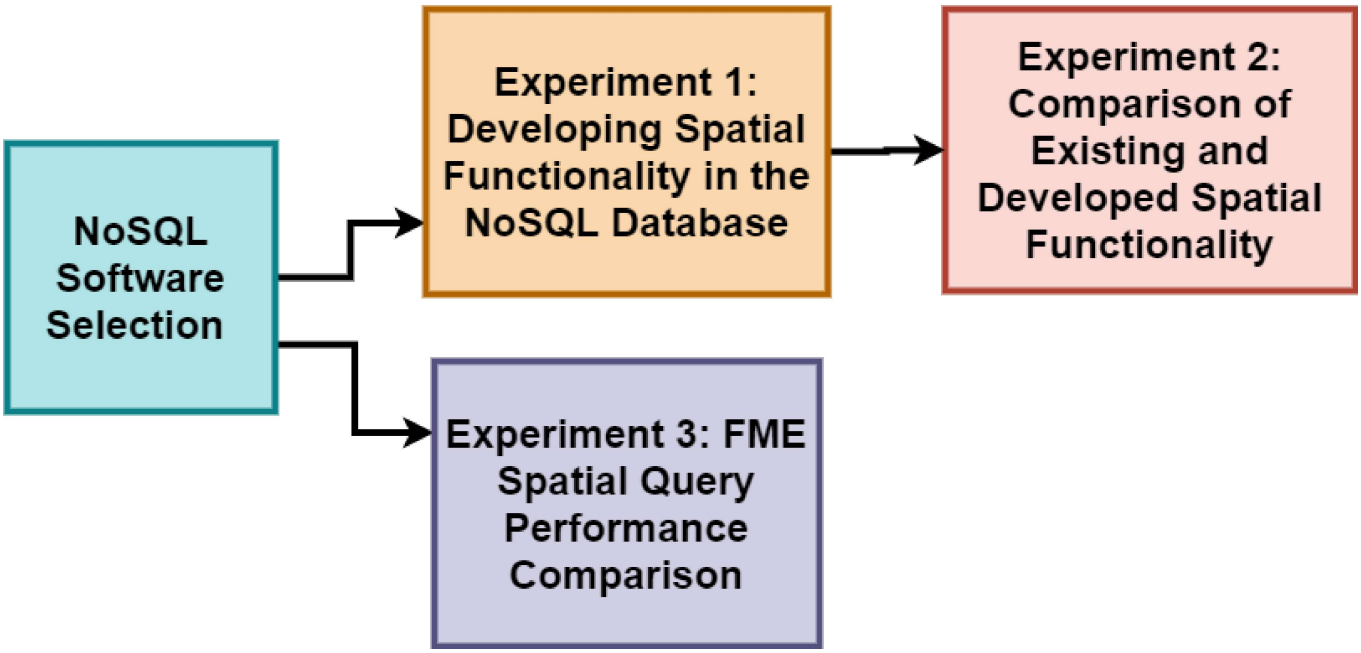


Figure 5 Hierarchical data flow diagram of NoSQL software selection and experiments. Colours represent the separate stages.

5.1 Test Environment

Table 4 displays all the software and hardware used in each Experiment. FME was used to manage and transform as well as insert data into the databases. Data was also visualised and transformed in QGIS, and the code was developed and tested in the integrated development environment, Atom. Selection was based on the researcher's preference.

Table 4 Test environment software and hardware.

Software						Hardware
Version						
Atom	QGIS	FME	PostgreSQL	PostGIS	MongoDB	
1.58.0	3.20.2	2021.1.0.1	12.8	3.1.3	3.6	Make: Dell Processor: Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz RAM: 16.0 GB System type: 64-bit processor

5.2 Data Creation

This section details how the research objective of creating a 2D and 3D dataset for benchmarking was achieved.

The research used real vector data. Polygons of Building Heights and linestring representing the UK's Water Network was Sourced from the Ordnance Survey Collection on Digimap. The data was not checked for outliers; however, the OS VectorMap Local Product Guide state the quality control checks of the data, which the researcher relied on (Ordanance Survey, 2018).

FME was the primary data management tool. Figure 7 displays an example of a workbench used to create the datasets outputted in Table 5.

Apart from dataset I in Table 5, each experiment used five datasets incremental in size, which were split using the data sampler in FME. The majority of established research in Section 3.5 used three datasets. Five datasets were chosen to increase the focus on the effect of incremental loading on performance. Aside from the 1 million points generated in QGIS that imitated Pietroń (2019), the count of geometries was constrained to data download restrictions available on Digimap or what the hardware could process using QGIS. For example, four 5 km by 5 km tiles is the reason for dataset F5's size (Table 5). Three functionalities were used in QGIS, 'generate random points' (datasets A and B), 'Convert Polygons to Points' to turn building height data into points (dataset H) and 'Set Z value' to turn 2D geometries into 3D (datasets E, H, and I in Table 5).

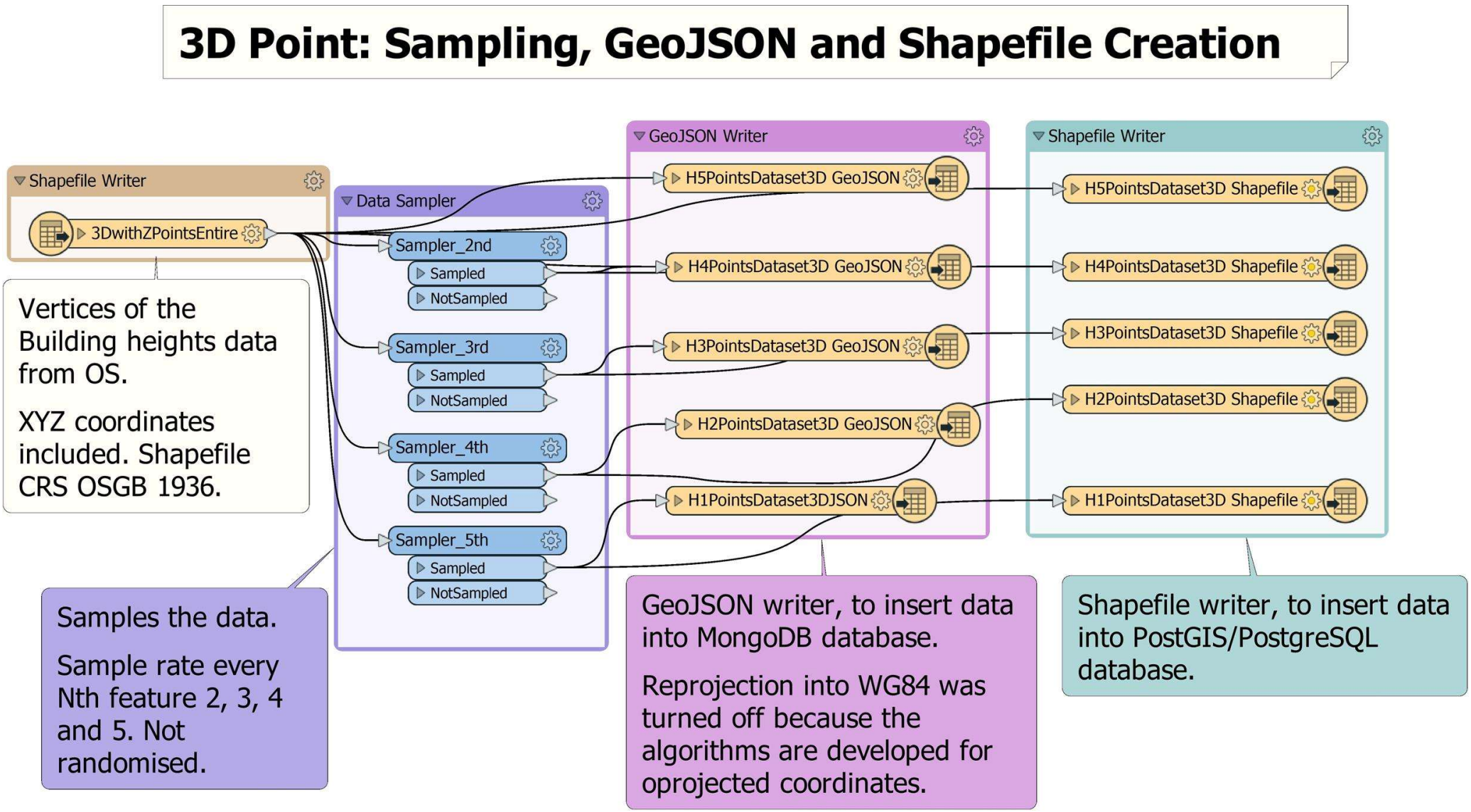


Figure 6 FME workbench to create dataset H1.

Table 5 Summary information of datsets.

Summary of Created Datasets									
Datasets		Geometry Count							Dataset Size (in MongoDB)
EPSG:2770 No Index	EPSG:4326 Indexed	Point	Linestring	Polygon	Polyhedral	3D Point	3D Linestring	Total	
A1	B1	200000						200000	23.8 MB
A2	B2	250000						250000	29.8 MB
A3	B3	333333						333333	39.7 MB
A4	B4	500000						500000	59.6 MB
A5	B5	1000000						1000000	119.1 MB
C1	D1	200000	87798	78354				366152	115.4 MB
C2	D2	250000	109748	97943				457691	144.6 MB
C3	D3	333333	146331	130590				610254	192.5 MB
C4	D4	500000	219497	195886				915383	289.1 MB
C5	D5	1000000	438994	391772				1830766	395.6 MB
E1					78354			78354	131.9 MB
E2					97943			97943	82.4 MB
E3					130590			130590	109.9 MB
E4					195886			195886	164.9 MB
E5					391772			391772	164.9 MB
F1				78354				78354	131.9 MB
F2				97943				97943	82.4 MB
F3				130590				130590	109.9 MB
F4				195886				195886	164.9 MB
F5				391772				391772	164.9 MB
G1							78354	78354	131.9 MB
G2							97943	97943	82.4 MB
G3							130590	130590	109.9 MB
G4							195886	195886	164.9 MB
G5							391772	391772	164.9 MB
H1						697664		697664	162.3 MB
H2						872080		872080	270.4 MB
H3						1162773		1162773	3500.4 MB
H4						1744160		1744160	405.6 MB
H5						3488320		3488320	811.3 MB
I1					10			10	20.0 KB
I2					50			50	20.0 KB
I3					100			100	20.0 KB
I4					500			500	20.0 KB
I5					1000			1000	28.0 KB
I6					5000			5000	64.0 KB
I7					10000			10000	120.0 KB
									Area
P1				1					41837275 m^2
P2				1					111055954 m^2
P3				1					182132487 m^2

5.3 Experiment 1: Extending Spatial Functionality in MongoDB

Experiment 1 aims to answer the research question of if a NoSQL DBMS can be enabled to support 2D and 3D spatial functionality similar to an RDBMS. Figure 8 outlines the methodology of the experiment.

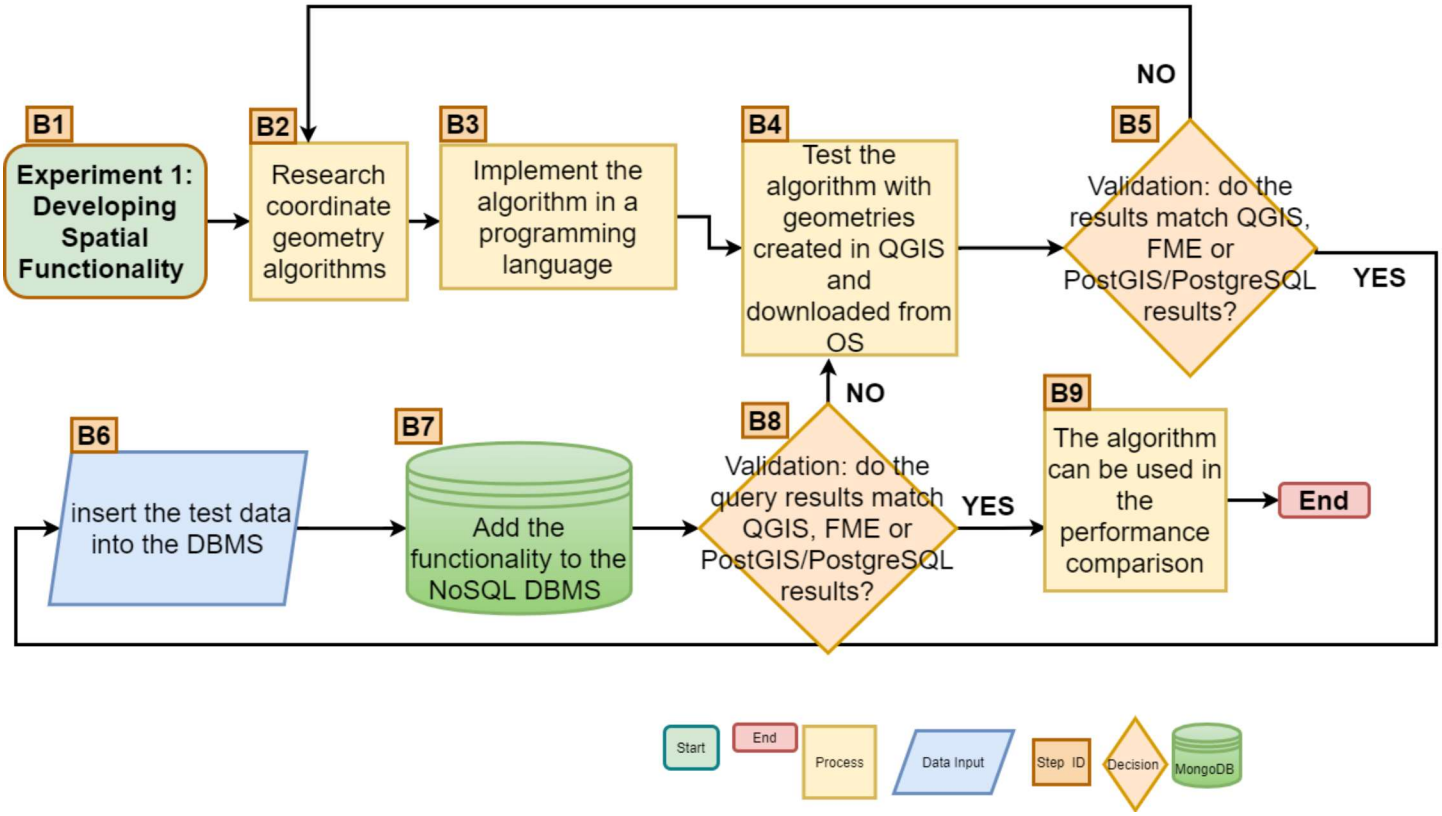
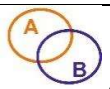
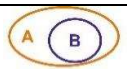


Figure 7 Experiment 1 flow diagram.

5.3.1 Coordinate Geometry Algorithms

Step B2 in Figure 8 requires researching the algorithms of spatial operations. Table 3 shows that PostGIS has over 1000 spatial operators, the production of that many operators is not feasible in this project. The researcher identified the overlapping spatial operations in the OGC Simple Features documentation, PostGIS documentation.FME documentation (FME, 2021), Section 5 Computational Geometry in Rigaux, et al. (2002) and Section 1 Basic Geometric Operations by Xiao (2016). Table 6 displays the results. A list of ideal spatial operators to develop.

Table 6 Desired spatial operators

Ideal list of Spatial Operators						
Spatial Operators		Descriptions and Illustrations	Completion	Algorithms adapted from or sourced from	Application of Example	Limitations and assumptions (All only projected coordinates)
1	2D distance Projected	Distance between two geometries in metres, XY coordinates.		Xiao (2016)	Distance between two resturants on map.	Only Points and as the crow flys
2	3D Distance	Distance between two geometries in metres, XYZ coordinates.		Xiao (2016)	Calculating distance between two rooftops.	Only Points
3	Area	Size of the surface in m^2.		Xiao (2016) and Page (2011)	Area of greenspace.	Coordinates must
4	2D Perimeter/Length/ 3D Perimeter/ 3D Length	Distance in metres around the geometry or its length.		Adapted from 2D Distance	Length of a road or perimeter of a building.	
5	Volume	Quantity of 3D space in m^3		Adapted from Area	Volume of a building.	Maximum volume estimate
6	Disjoint	The boundary and interior of both geometries do not touch, which returns a boolean data type.		Adapt Within	Identify shops not within a town.	
7	Equals	Geometries have the same boundary and same interior, which returns a boolean data type.			Identify if it is the same building.	
8	Touches	Geometry interiors do not intersect but boundaries do, which returns a boolean data type.			Finding which countries border each other.	
9	Crosses	 Returns a boolean data type		Cormen, et al., (2009)	Which streams intersects.	
10	Overlaps	 A and B intersect but are not equal or contains. Returns a boolean data type.			If a proposed running route overlaps major roads.	
11	Contains	 A contains B. Returns a boolean data type.		Adapt Within operator	Identify restaurants within a particular area.	
12	Within	 A is within B. Returns a boolean data type.		Xiao (2016) and Rosetta Code (2010)	Identify if restaurants are within a particular area.	Only Points within a Polygon
Key		Algorithm was attempted and failed or not attempted but the theory is known | Algorithm was not attempted | Algorithm worked | geometry B | geometry A |				

The table will be explained in more detail in Section 6

5.3.2 Example of Alogrithm

Figure 9 displays an example of a coordinate geometry algorithm in JavaScript that will be embedded into MongoDB function in Figure10

Polygon Area Algorithm

```
//Code sourced from Darel Rex Finley (2011) Algorithm to find the area of a polygon [Source //code]
https://www.mathopenref.com/coordpolygonarea2.html

//Inputs

//var X=array of x coordinates;

//var Y=array of Y coordinates;

//var verticeCount= length of coordinate array;

1.  function area (X ,Y , verticeCount){
2.          var area=0;
3.          var j= verticeCount-1;
4.          for (i=0; i< verticeCount; i++)
5.                  { area +=  (X[j]+X[i]) * (Y[j]-Y[i]);
6.                  j = i;
7.                          }
8.                  return area/2;
9.                  }
```

Figure 8 Code for polygon area.

5.3.3 Algorithm Testing and Validation

Step B3 and B4 in Figure 8 were developed and tested in Atom, functions in MongoDB are written in JavaScript. The mathematical algorithms were translated into JavaScript and tested using shapefiles created in QGIS. All created functionalities were for projected coordinates not geographic because the NoSQL databases researched did not support geometries in projected coordinates. Figure 10 shows an example of how data was validated in FME by creating an attribute with the results of the equivalent spatial operation or by running the equivalent query in PostGIS. If the results matched to two decimal places, they were considered valid. Steps B6–B8 in Figure 8 is outlined in Figure 10.

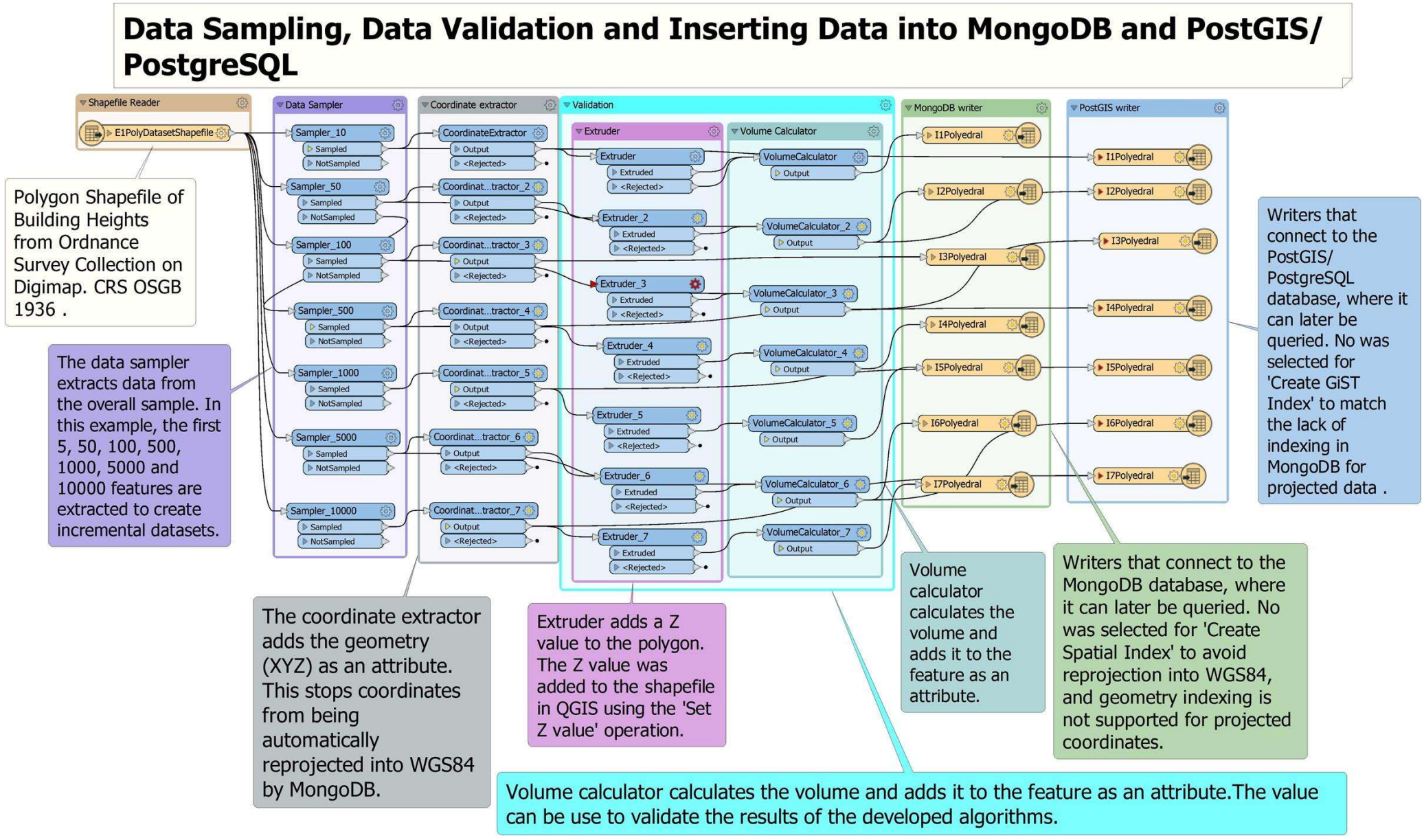


Figure 9 FME work bench, data validation and insertion into databases.

5.4 Experiment 2: Performance Comparison of Existing Spatial Operators in PostGIS/PostgreSQL and MongoDB

Experiment 2 is designed to answer the research questions of whether the main supported equivalent spatial functions in a NoSQL and RDBMS have similar query performance. Likewise, if the researcher created functionality has a similar performance. Figure 11 presents the process.

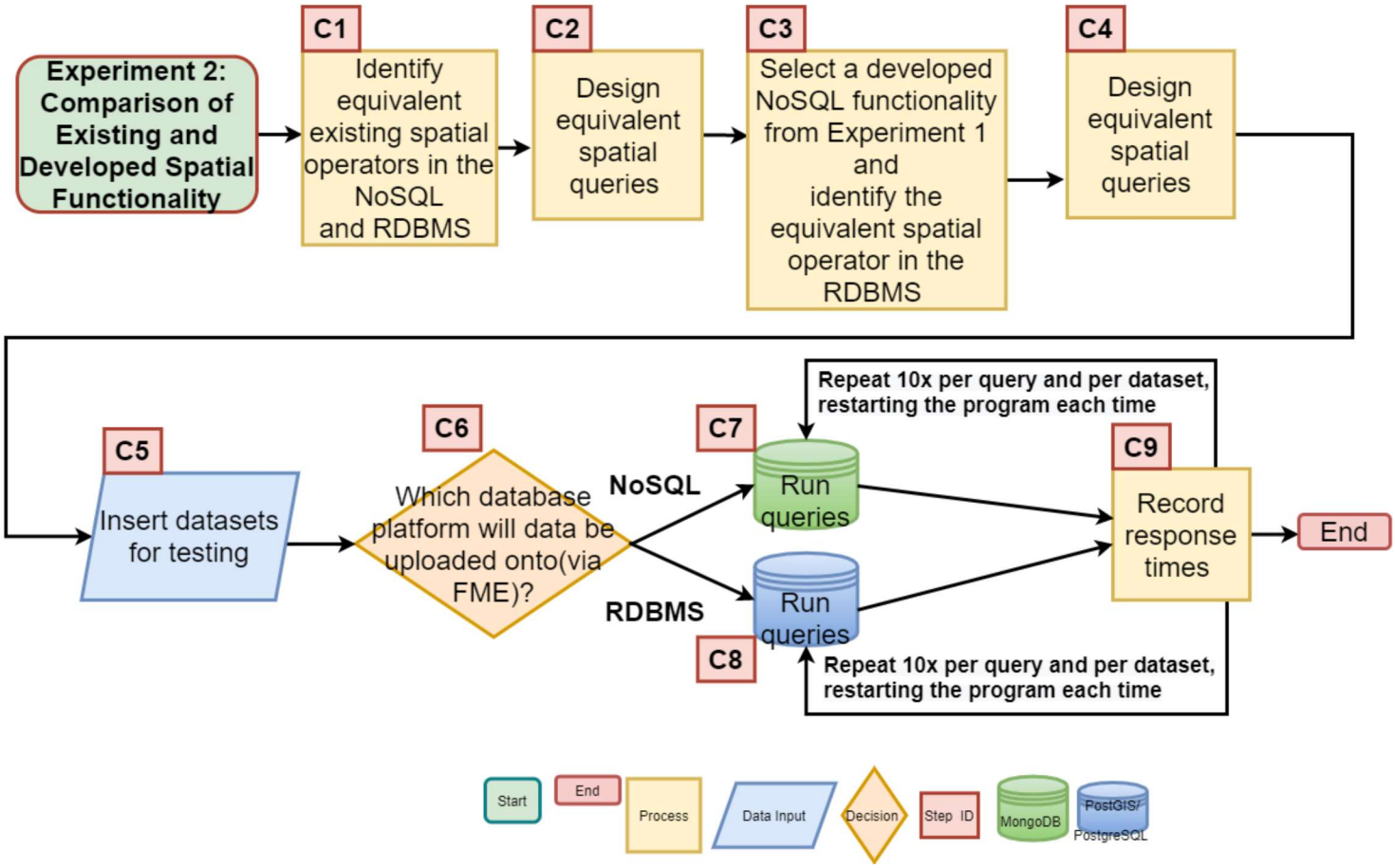


Figure 10 Flow diagram for Experiment 2.

5.4.1 Experiment 2 Methodology

Steps C1 and C2 in Figure 11 required researching the PostGIS and MongoDB documentation to identify equivalent spatial operators for comparison. (appendix). Step C3 in Figure 11 is dependent on the results of Experiment 1. If no functionality is developed it cannot be carried out. Similar to Experiment 1 steps C5 and C6 in Figure 11: the method of data insertion into the database is shown in the writers section in Figure 10.

Step C7 and C8 in Figure 11 were carried out in the command shell of each platform. The explain method was added to the query in MongoDB see Figure 13, whereas the '\timing' command timed queries in PostGIS. Each query ran 10 times, mirroring the previous studies in Table 1. The average of the response time in milliseconds were recorded and the standard deviation.

5.4.2 Experiment 2 Queries

Query of existing spatial operators:

1. Points, linestrings and polygons within a polygon query using dataset C or D in Table 5 and repeated using three polygons as filters, dataset P.

Developed spatial operators benchmarking queries:

2. Volume calculation on datasets E
3. Area calculation on dataset F
4. Point in polygon query using dataset A or B and polygons in dataset P functioning as a filter.

Figures 12 and 13 Show an example of an equivalent volume calculation query in MongoDB and PostGIS/PostgreSQL.

SQL Volume Calculation Query

```
nction (geojson,vertices) {var x = geojson.x[0], y = geojson.y[0]; var inside = 0; for (var i
= 0, j "
```

Figure 11 Volume query in SQL. See appendix for explanations.

MongoDB NoSQL Volume Calculation Query

```
Db.datasetI1([
{$project:
{ Volume: {$function: {
body: "function (geojson){Algorithm } "
,
args: ["$geometry"],
lang: 'js'}
}
}
}]).explain("executionStats")
```

Figure 12 Volume query in NoSQL . See appendix for explanations.

5.5 Experiment 3: FME Performance Comparison of Existing Spatial Operators in PostGIS/ PostgreSQL and MongoDB

Experiment 3 was formed as a result of research into extending the spatial capabilities of NoSQL databases similar to Experiment 1. However, it uses the middleware FME to extend spatial capabilities. The process is summarised in Figure 14.

5.5.1 Experiment 3: Methodology

Figure 15 details the entire process of Experiment 3. It parallels Experiment 2 in many aspects, for instance the method of data insertion; however, if differs by carrying out the spatial operation outside of the database: queries were ran inside of FME not a command shell and required the software to read in the geometries from the database. Another difference is that the CPU of the query as well as the execution time was recorded.

FME Query:

1. Point in polygon using dataset A and B and polygons in dataset P functioning as a filter.

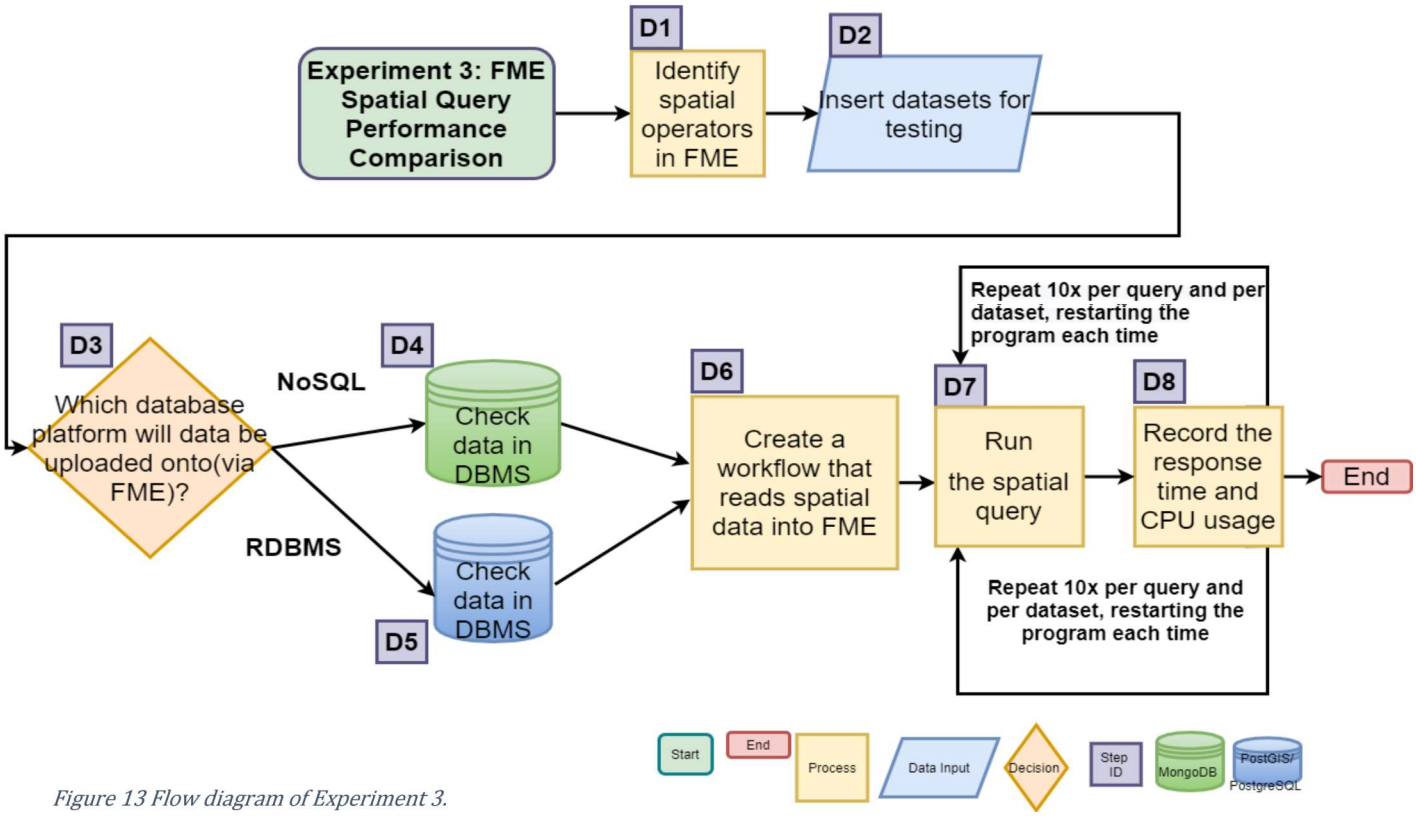


Figure 13 Flow diagram of Experiment 3.

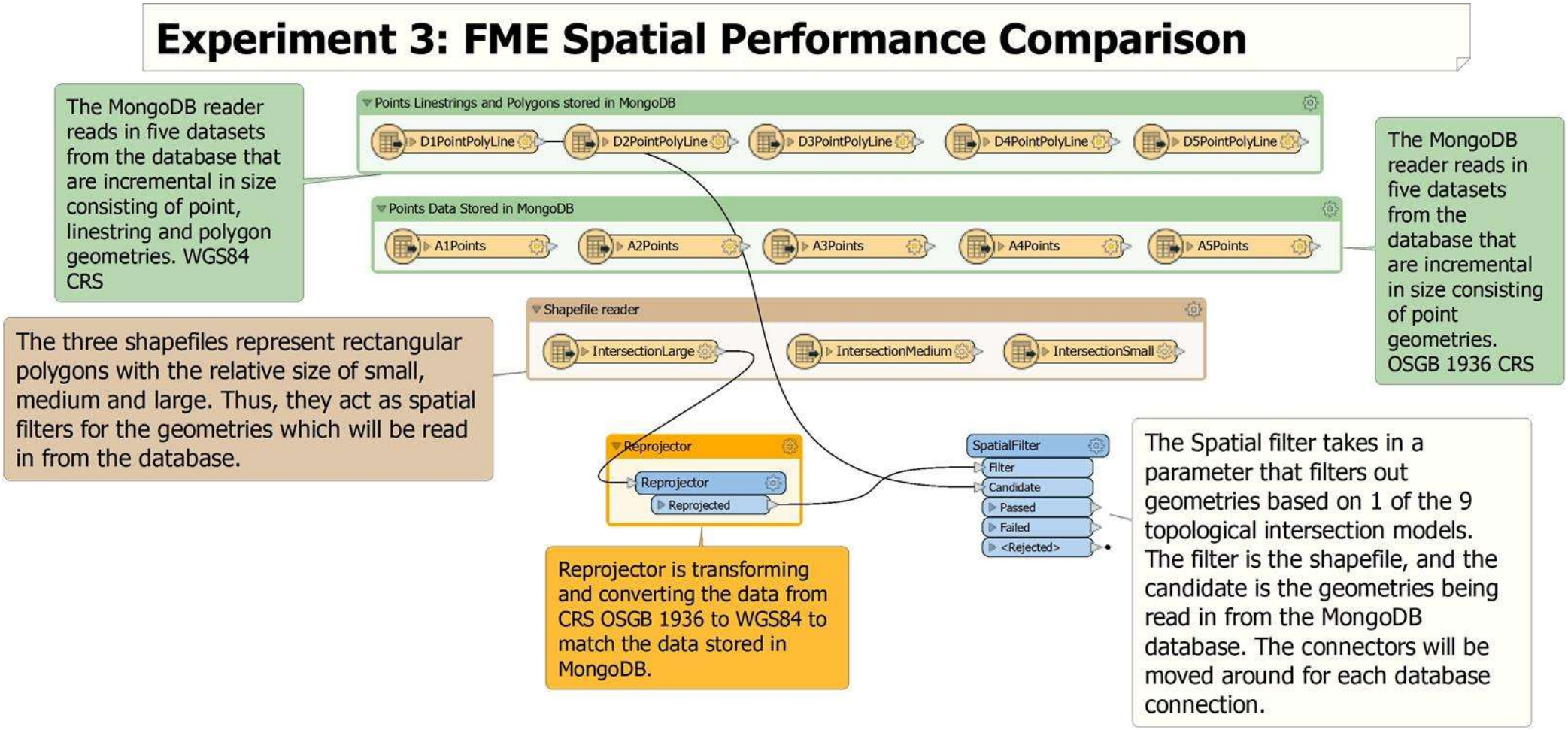


Figure 14 Experiment 3 FME workbench

6 Results

This Section details the overall results of the spatial performance comparisons between MongoDB and PostGIS/PostgreSQL.

6.1 Experiment 1 Results

The purpose of Experiment 1 was to investigate if it is possible to extend the NoSQL database to have similar 2D and 3D functionalities as a RDBMS, for example, PostGIS/PostgreSQL. The results of Experiment 1 are presented in Section 5.3, Table 6. Out of the 12 operators identified as suitable for development, six were created, and returned accurate results. Most of the spatial operators developed related to euclidean geometry: area and distance. Implementing topological operators were more complex, and fewer sources were available to support the creation, evident in the colour coding and sources column in Table 6. The developed operators were validated at two stages, and all were accurate to two decimal places when compared to equivalent results in FME or QGIS. Therefore, it is possible to extend the spatial functionality of the NoSQL database.

6.2 Experiment 2 Results

6.2.1 Existing Spatial Operator: Within Query

The aim of Experiment 2 was to test the performance of existing and developed spatial operators in NoSQL, to existing operators in a RDBMS and evaluate if similar performances can be achieved. Figure 16 illustrates an example of the Within query.

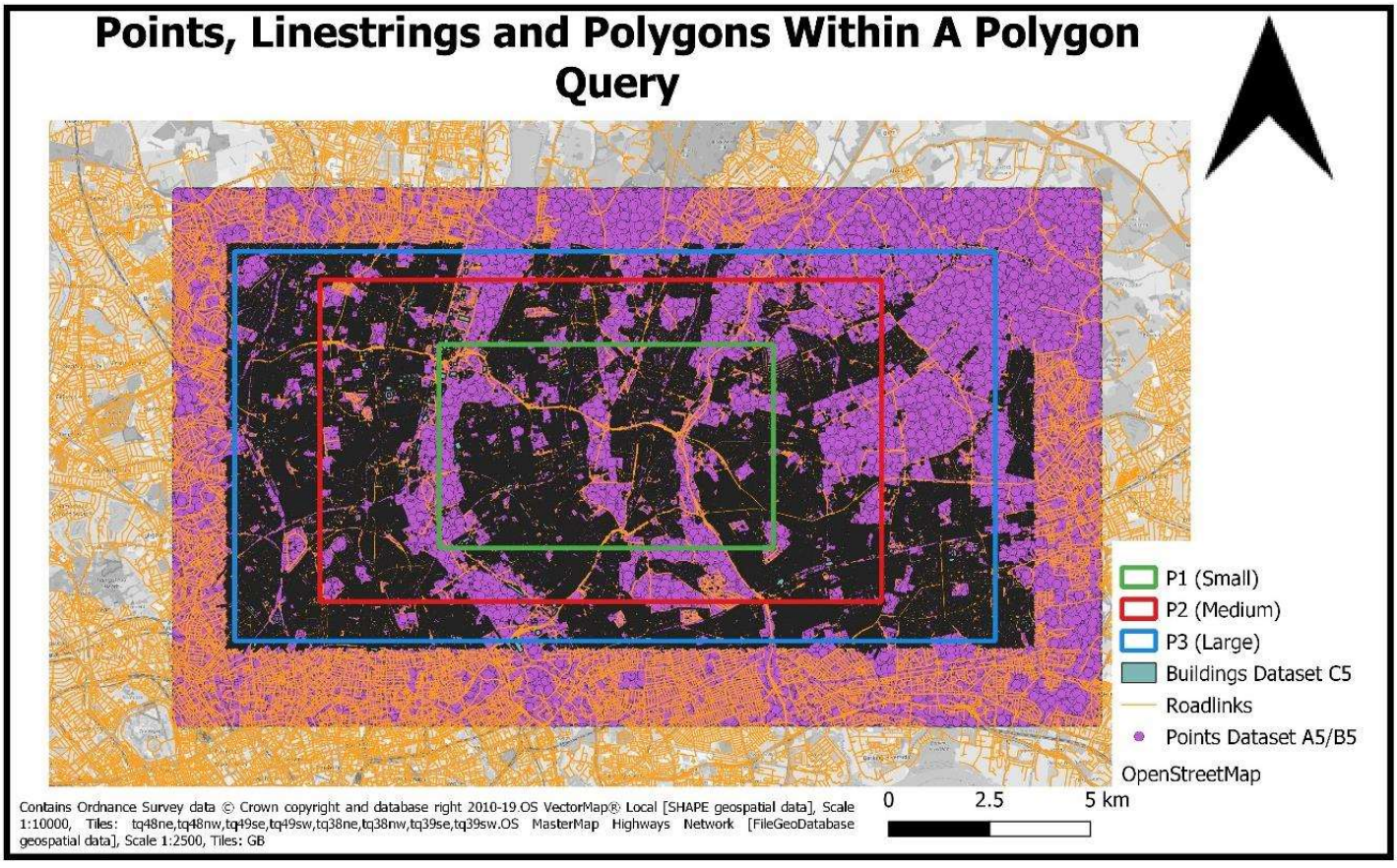
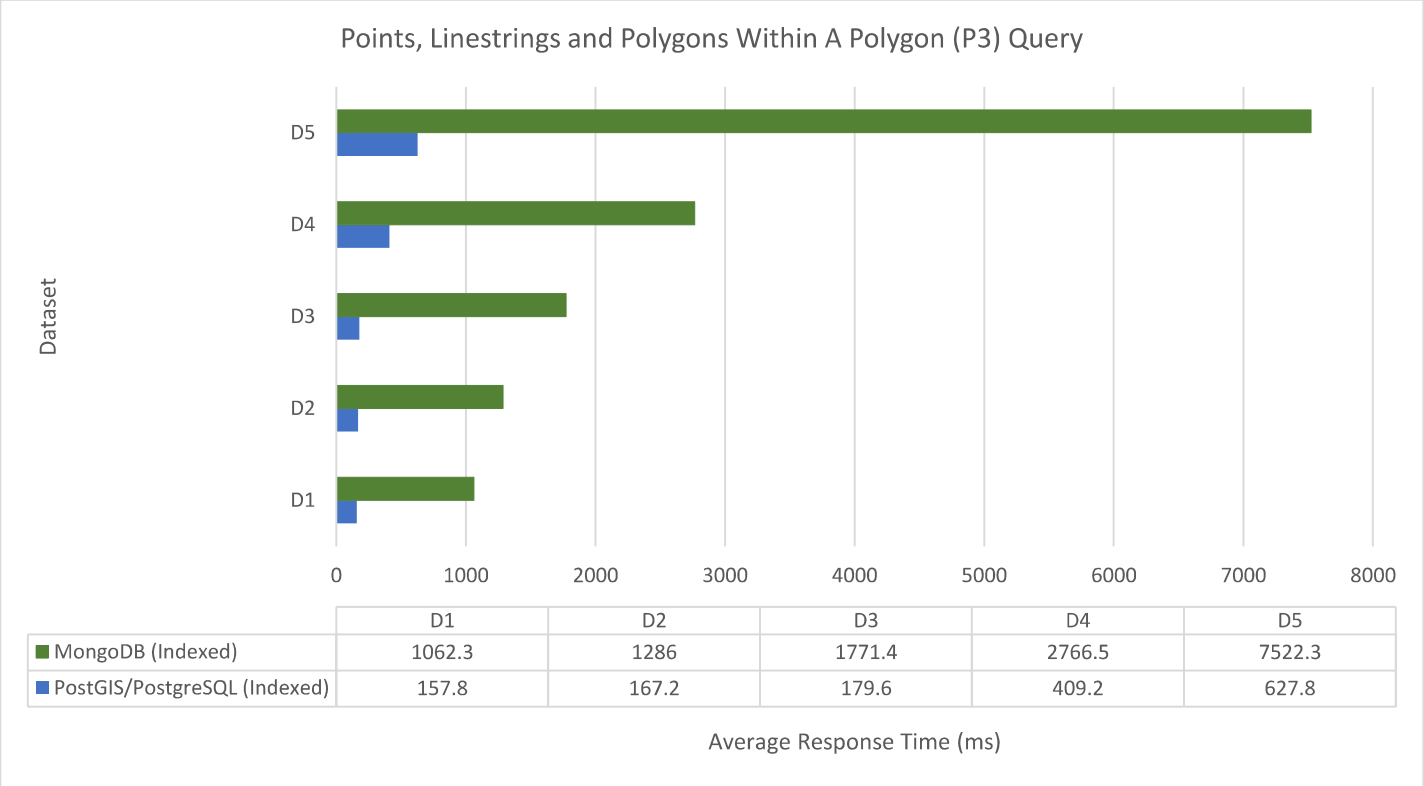


Figure 15 Map of the Within query. Generated in QGIS.

The Within query was executed 300 times: repeated for three polygons and the five datasets. Figure 17 shows an example of the average response time of the queried using polygon P3 (returning the most geometries). It is clear that PostGIS returns queries faster than MongoDB, this trend is the same for the majority of iterations. Query speed ranged from being 1.3-11 times faster than MongoDB including testing on polygons P1 and P2. In this example the existing operators did not have a similar performance in both DBMSs.



	D1	D2	D3	D4	D5
MongoDB (Indexed)	1062.3	1286	1771.4	2766.5	7522.3
PostGIS/PostgreSQL (Indexed)	157.8	167.2	179.6	409.2	627.8

Figure 16 Results of Within query benchmark.

6.2.2 Developed Spatial Operators: Area and Volume Query

Initially dataset E (Table 5) was created to test the volume operator. Dataset E1 had the smallest sample size of 78,354 polyhedrons. When the query ran in PostGIS, it failed to compute a result after 20 minutes. Therefore, the test was deemed a failure. In MongoDB, the volume query took on average 1316 ms for dataset E1 and 6200 ms for E5, whereas it failed in PostGIS. As a result, a smaller dataset assigned as 'I 'was created to test the volume and area. Figure 19 illustrates the buildings used in the polyhedral dataset E and I and when in 2D, it is the polygon dataset F.

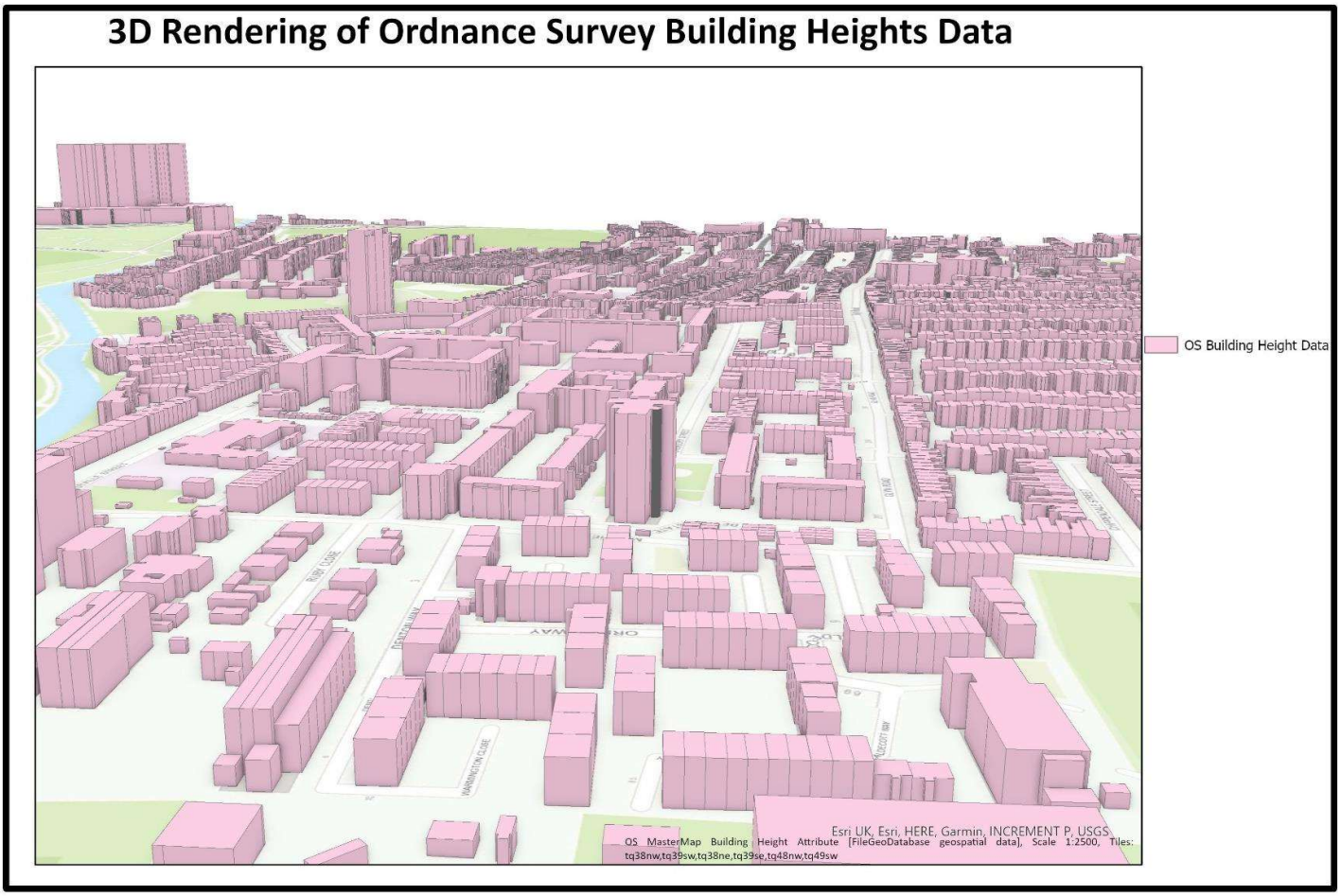
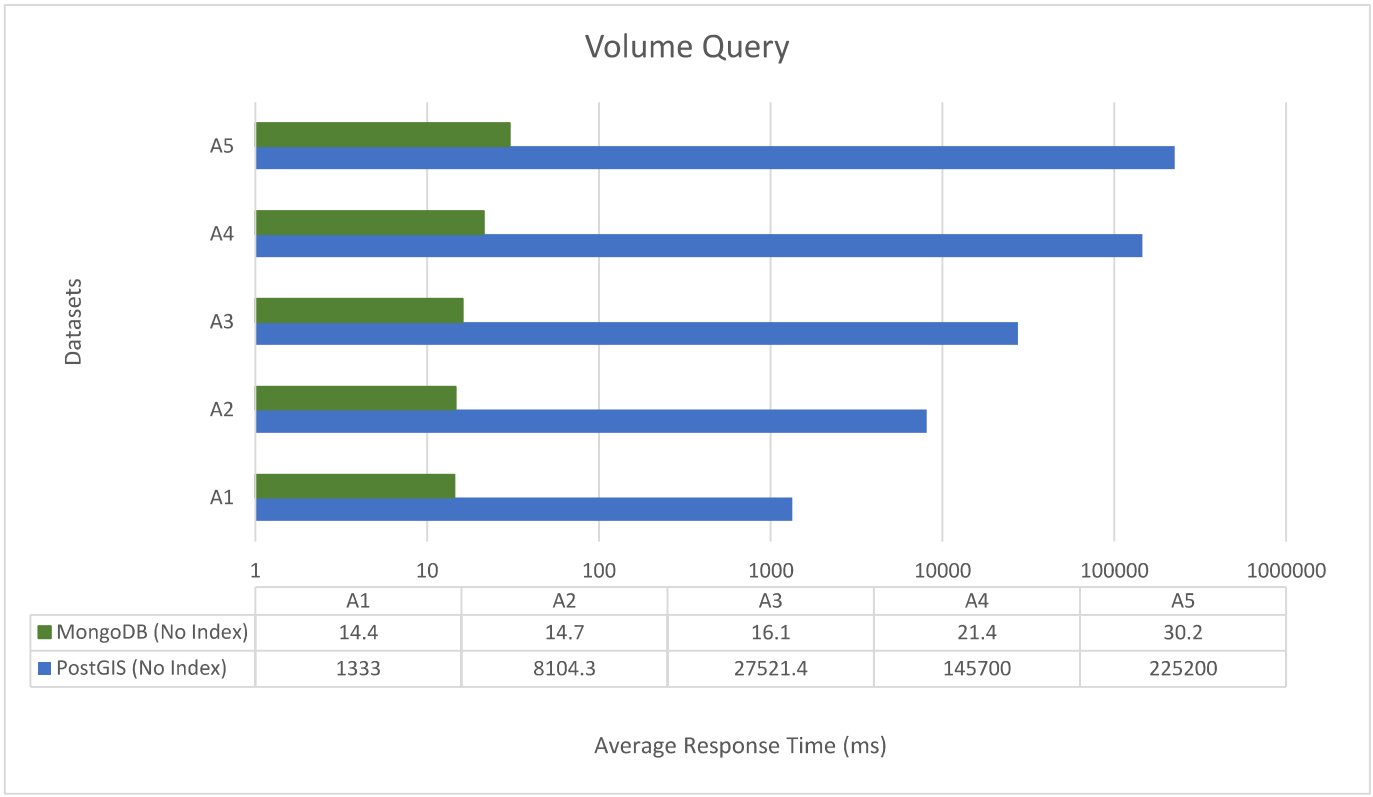


Figure 17 OS Building Heights map. Generated in QGIS.

There was a surprising and stark difference in the Volume experiment. Figure 18 illustrates the results. The significant time difference required a log scale for the horizontal axis. When compared in 50 tests, MongoDB outperformed PostGIS at a rate of between 91-7456 times quicker. MongoDB returned queries within milliseconds range of 14.4-30.2 whereas PostGIS took up to 3 minutes. The query showed an example of the developed functionality outperforming an existing one.



	A1	A2	A3	A4	A5
MongoDB (No Index)	14.4	14.7	16.1	21.4	30.2
PostGIS (No Index)	1333	8104.3	27521.4	145700	225200

Figure 18 Volume query results.

The results of the Area query were varied. The most interesting aspect of Figure 50 is the transition in query response time between queries on I5 and I6. MongoDB responded quicker when the dataset was smaller, however, as the dataset increased the response time grew significantly. Conversely, over the 70 tests, PostGIS showed no overall pattern but remained in a range of between 42.1-59.4 ms. Table 7 displays the standard deviation of average response times, it gives an indication of why PostGIS results were diverse. The query response times had a greater spread, particularly the I1 and I6 which influenced the average.

	I1	I2	I3	I4	I5	I6	I7
MongoDB (No Index)	14.5	15.3	14.9	20.7	27.2	83	157.7
PostGIS (No Index)	59.4	48.4	49.8	42.1	44	42.5	45.5

Figure 19 Area query results.

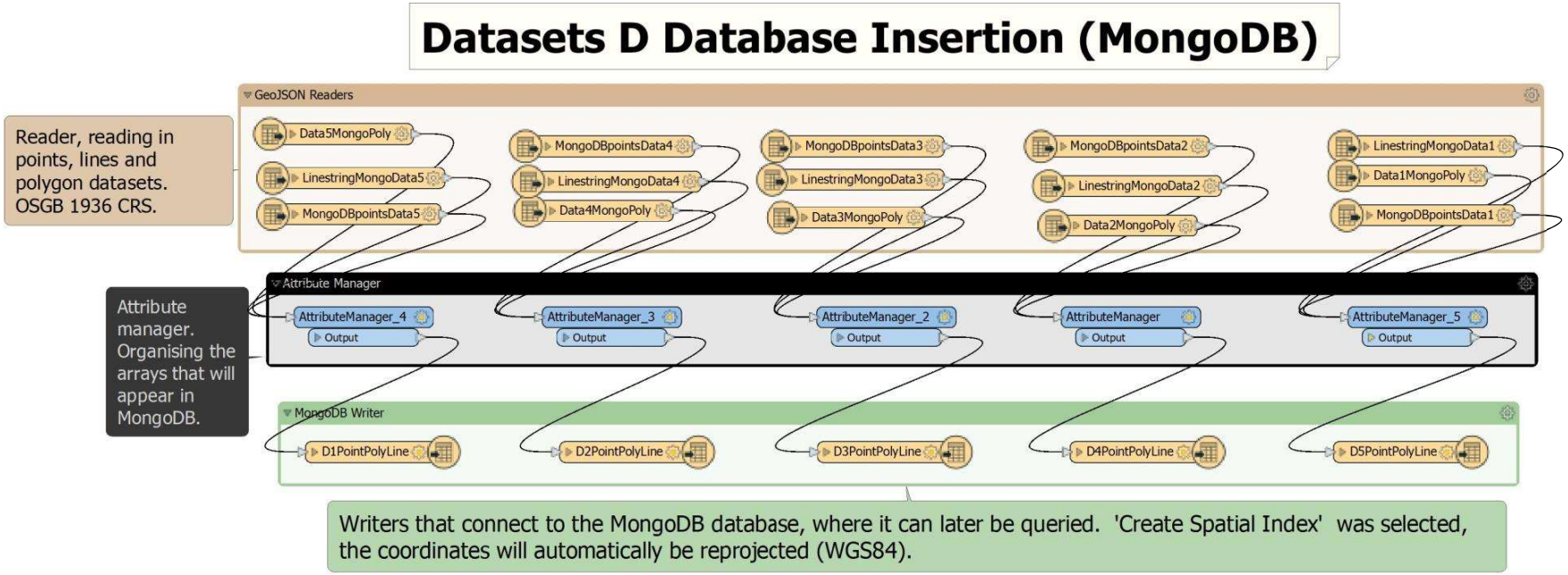
Table 7 Standard Deviation of Area Query Response Times

Standard Deviation of Area Query Response Times							
Datasets	I1	I2	I3	I4	I5	I6	I7
PostGIS Standard Deviation of Response times (ms)	17.60682	1.197219	0.674949	16.48198	12.17465	2.173067	6.276057
MongoDB Standard Deviation of Response times (ms)	0.527046	1.636392	1.197219	0.674949	1.135292	2.538591	8.577101

12 Appendices

Appendix A

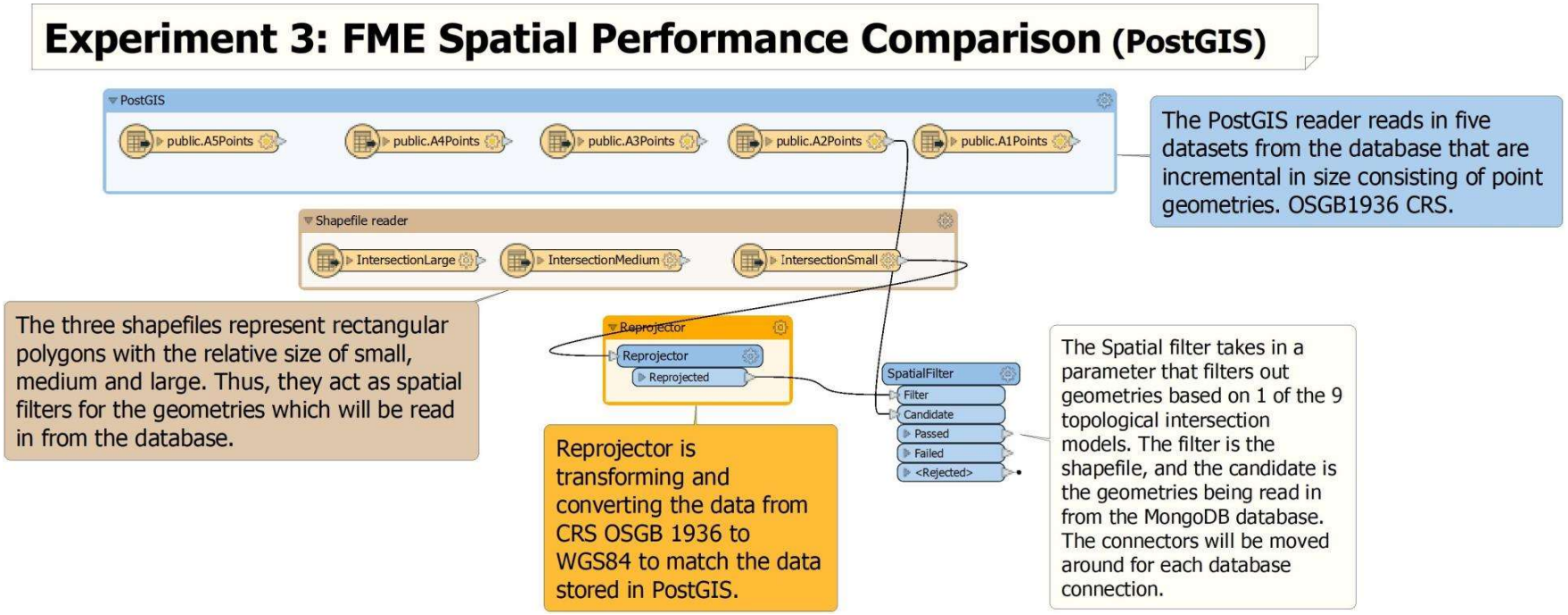
Datasets D Database Insertion (MongoDB)



Appendix A Datasets D Database Insertion (MongoDB) FME Workbench.

Appendix B

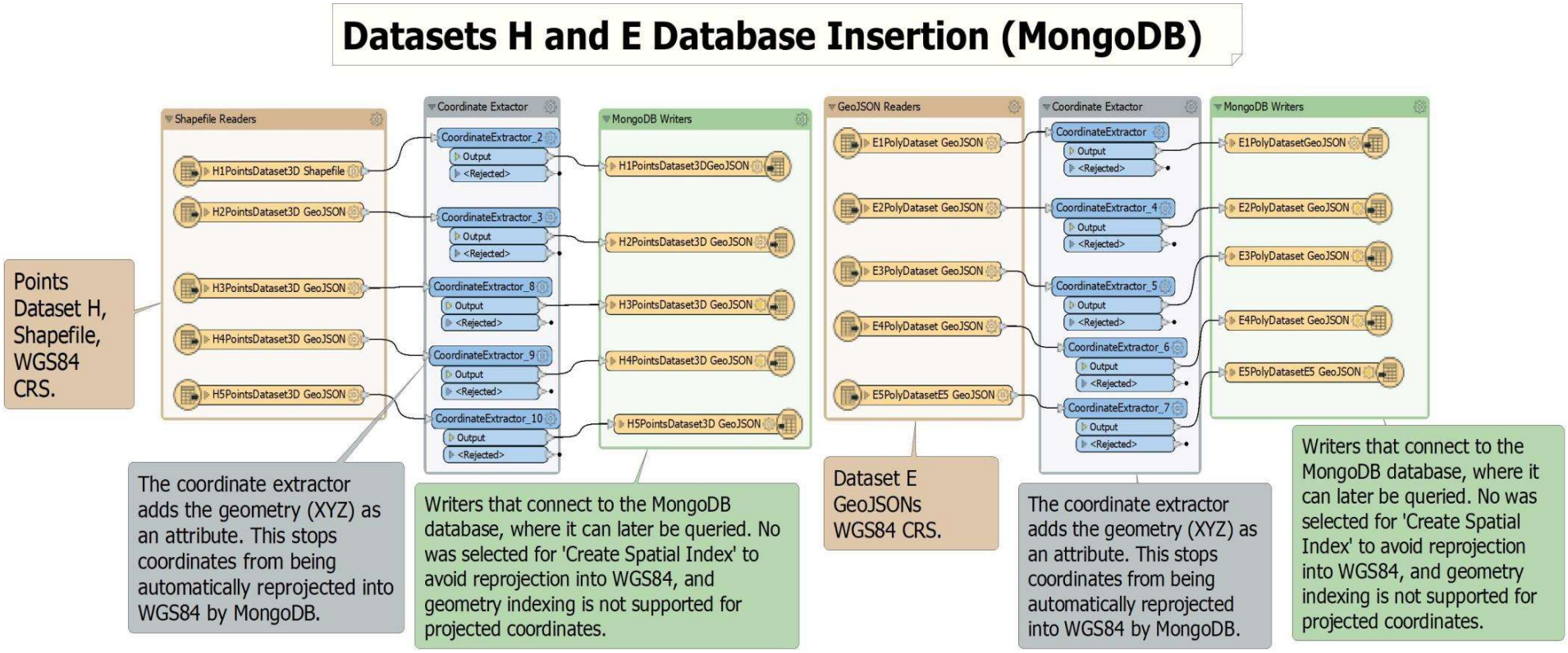
Experiment 3: FME Spatial Performance Comparison (MongoDB)



Appendix B Experiment 3: FME Spatial Performance Comparison (MongoDB) FME workbench.

Appendix C

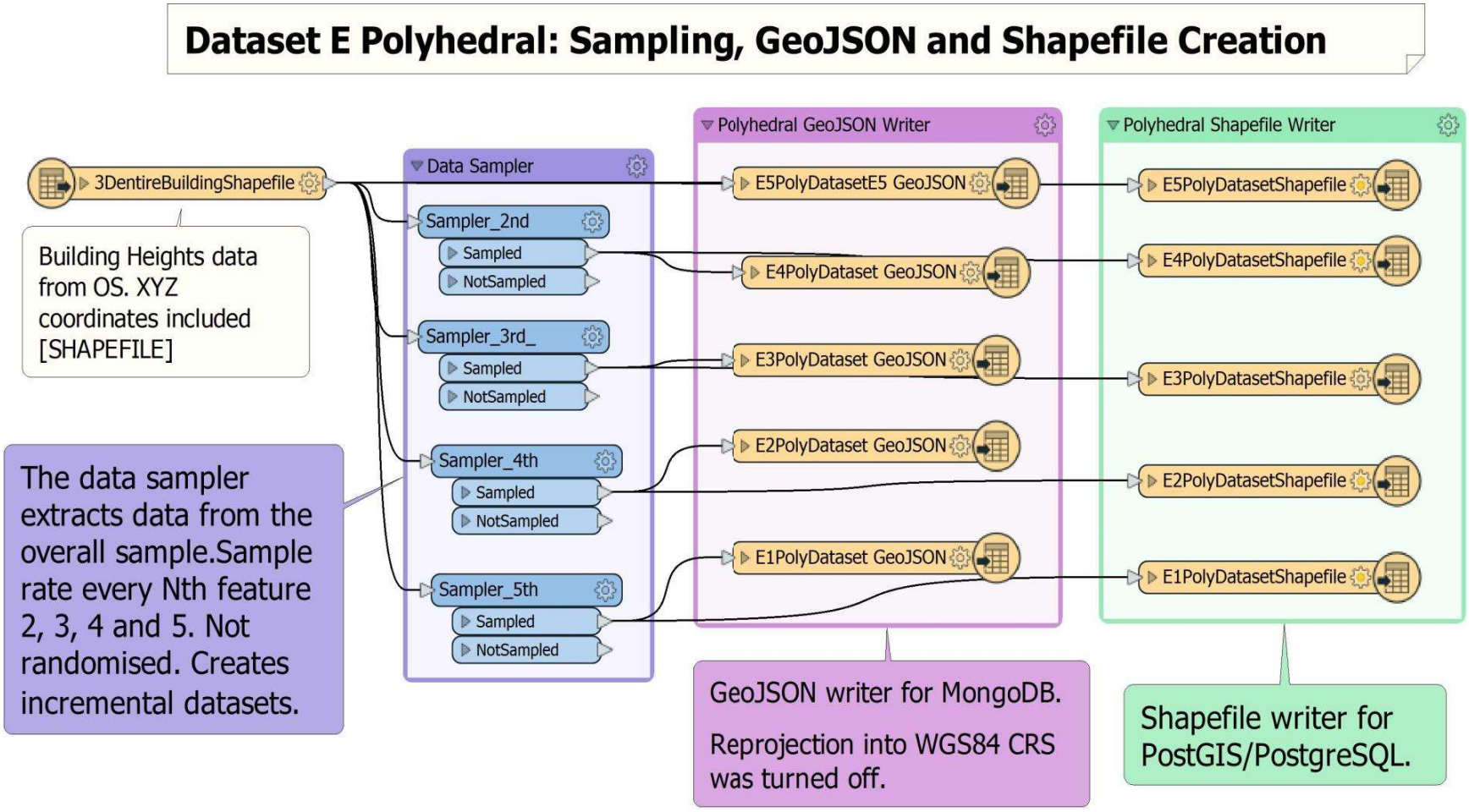
Datasets H and E Database Insertion (MongoDB)



Appendix C Experiment 3: FME Spatial Performance Comparison (MongoDB) FME workbench.

Appendix D

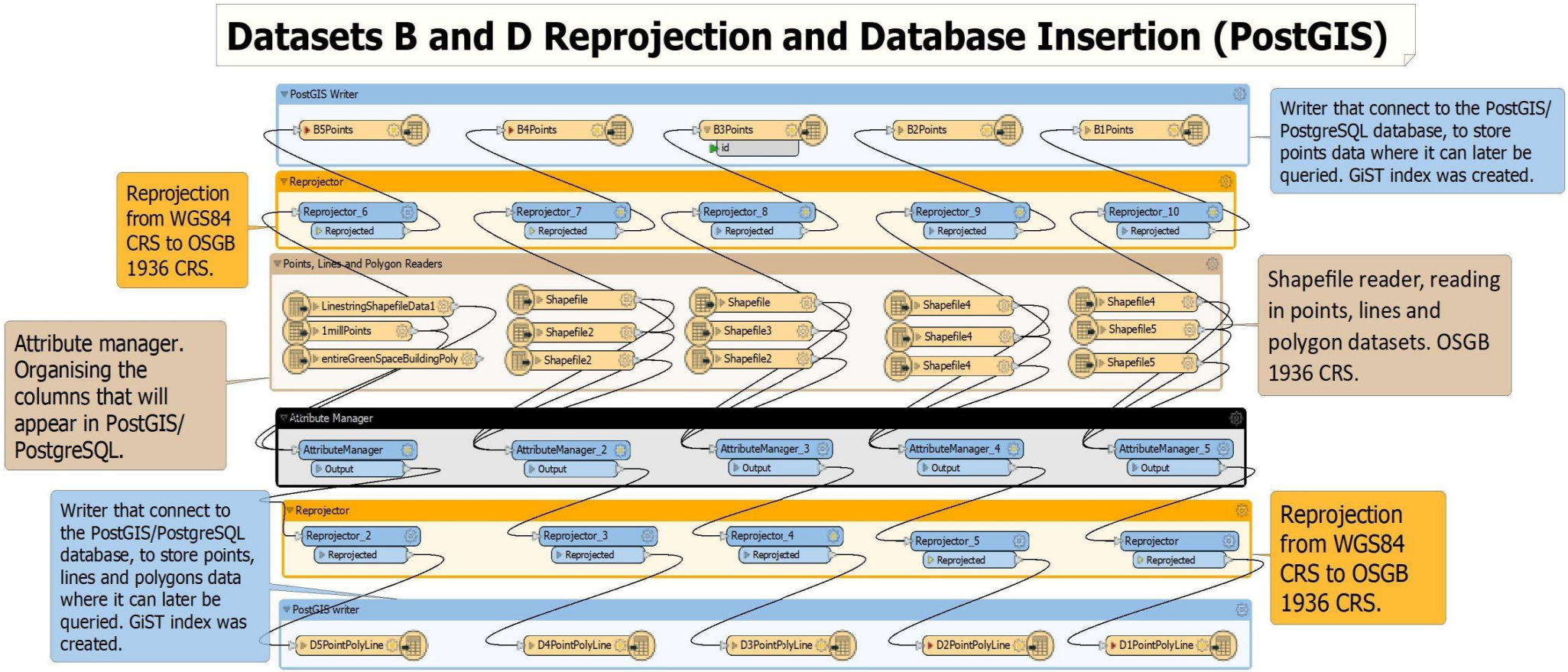
Dataset E Polyhedral: Sampling, GeoJSON and Shapefile Creation



Appendix D Dataset E Polyhedral: Sampling, GeoJSON and Shapefile Creation FME workbench

Appendix E

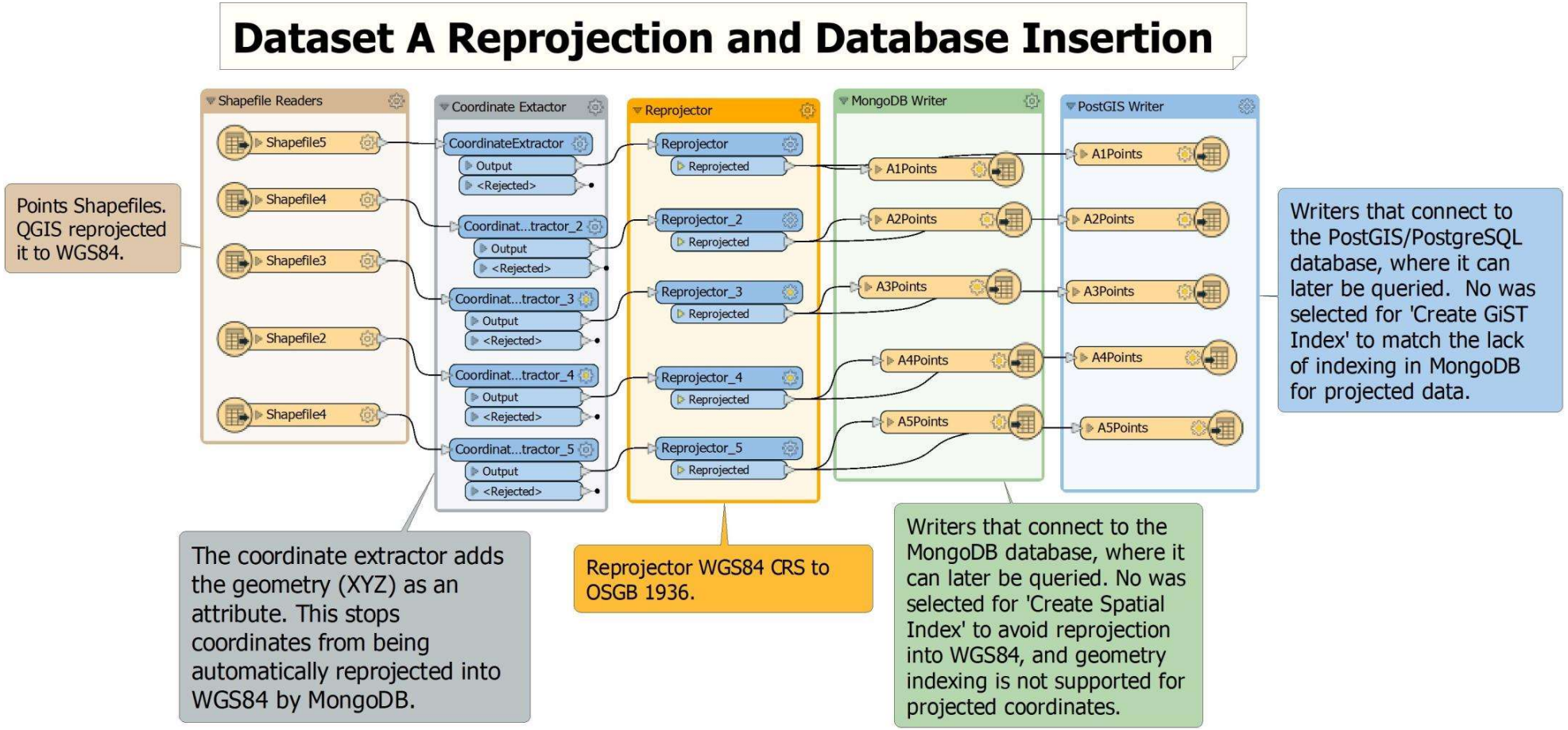
Datasets B and D Reprojection and Database Insertion (PostGIS)



Appendix E Datasets B and D Reprojection and Database Insertion (PostGIS) FME workbench.

Appendix F

Dataset A Reprojection and Database Insertion



Appendix F Dataset A Reprojection and Database Insertion FME workbench.

Appendix G

Equivalent Spatial Operators In MongoDB and PostGIS/PostgreSQL

Appendix G Spatial operators. Sourced from Pietroń(2019).

Equivalent Spatial Operators in MongoDB and PostGIS	
MongoDB	PostGIS
$geoIntersects	ST_INTERSECTS
$geoWithin	ST_WITHIN
$geoNear	ST_BUFFER+ST_INTESECTS
$near	ST_BUFFER+ST_INTESECTS

Appendix H

NoSQL and RDBMS (SQL) Commands

Appendix H Table summarising NoSQL and RDBMS (SQL) Commands. Data sourced from MongoDB (2021) & PostgreSQL (2021).

NoSQL and RDBMS (SQL) Commands			
MongoDB Statements	Definition	PostGIS/PostgreSQL (SQL) Statements	Definition
db.DatasetE1.find	Selects documents in collection 'DatasetE1'	SELECT	Selects data from a database
$geoWithin	Spatial operators returning documents within a specified filter	COUNT	Returns the number of rows that match a condition
coordinates: [[[-0.0286786, 51.5725885],...	Specifies the coordinates of the filter for a spatial operator	WHERE	Filters out records based on a specified condition
explain("executionStats")	Provide statistics about a performed query, e.g., execution time	FROM	Selects a specific table to retrieve data from
db.I2Polyhedral.aggregate	Performs an aggregation task; this is used before a function	ST_Within	Spatial operators returning documents within a specified filter
Volume: {$function: {}}	Defines a function in MongoDB part of the aggregation framework; in this instance, the value will return the volume	(geom,'SRID=27700 ;POLYGON((530186.6510982716 185161.1715020783 ...	Spatial filter which the user inputs the filters, SRID, its geometry type and its coordinates
args: ["$json_geometry"],	The argument that is passed into a function, in this instance, it is an array of coordinates, "$json_geometry"	ST_Volume(ST_Extrude (geom,0,0,relhmax))	It extrudes the geometry vertically based on the building height attribute 'relhmax', which Is the Z value, then calculates the volume
lang: 'js'	The language of the function, JavaScript	ST_Area	Returns polygon area
$project	Passes a document to the subsequent stage in the aggregation pipelin	ST_3DDistance	Finds the minimum cartesian distance between two 3D geometries
$match	Returns documents that match a specified condition		
$gt: 0	Greater than 0		
count: {$sum: 1}	Counts the number of documents returned that is greater than 1		

Appendix I

Within Query MongoDB

```
1.  db.DatasetE.find({
2.  geometry: {
3.  $geoWithin: {
4.  $geometry: {
5.  type: "Polygon" ,
6.  coordinates: [
7.  [
8.  [ -0.0286786, 51.5725885 ], [ -0.0286536, 51.5691836 ], [ -0.0286536, 51.5691836 ], [ -0.0286286, 51.5684325 ], [ -0.0286286, 51.5684325 ], [ -0.0220441, 51.5689332 ], [ -0.0237466, 51.5730642 ], [ -0.0286786, 51.5725885 ]]
9.  ]
10. }
11. }
12. }
13. }
14. ).explain("executionStats")
```

Appendix I Within Query MongoDB Syntax explanation, Appendix H

Appendix J

Within Query /Point in Polygon Query PostGIS/PostgreSQL (SQL)

Three different polygons were used P1, P2 and P3 (Table 5).

```
1. SELECT count(id) FROM public."A5Points" where ST_Within(geom,'SRID=27700 ;POLYGON((
2. 530186.6510982716 185161.1715020783 ,
3. 549053.7108801971 185161.1631545085 ,
4. 549053.7080037947 194811.5583172982 ,
5. 530186.6357099144 194811.5669819436 ,
6. 530186.6510982716 185161.1715020783
7. ))'::geometry)= true;
```

Appendix J Within Query /Point in Polygon Query PostGIS/PostgreSQL (SQL). Syntax explanation, Appendix H

Appendix K

Volume Query MongoDB

//Code adapted from Darel Rex Finley (2011) Algorithm to find the area of a polygon [Source //code] https://www.mathopenref.com/coordpolygonarea2.html

```
db.I2Polyhedral.aggregate([
{$project:
{ Volume: {$function: {
body: "function (geojson){
var xCoor = geojson.x; //loop through coordinate arrays, creating variables
var yCoor = geojson.y;
var zCoor = geojson.z;
var coordArray = geojson.x;
var  numPoints= coordArray.length;var X= []; var Y =[]; var Z =0;
 for (let i = 0; i < coordArray.length; i++)
{// create a list of coordinates
(X.push(xCoor[i]),
 Y.push(yCoor[i]),
 Z+=zCoor[i] )
}
Z=Z/numPoints;
var area = 0;
 var j; j = numPoints-1;// add the sum of cross multiplying the vertices offset by one then divide by two and multiple by the average height,e.g.,(x1*y2)
 for (vark =0; k<numPoints; k++) {
area +=  (X[j]+X[k]) * (Y[j]-Y[k]); j = k; }
 return (Math.abs(area/2))*Z; } " ,
args: ["$json_geometry"],
lang: 'js'}
}
}
}]).explain("executionStats")
```

Appendix K Volume Query MongoDB. Syntax explanation, Appendix H.

Appendix L

Volume Query PostGIS/PostgreSQL (SQL)

```
SELECT ST_Volume(ST_Extrude(geom,0,0,relhmax )) FROM public."I1Polyedral";
```

Appendix L Volume Query PostGIS/PostgreSQL (SQL). Syntax explanation, Appendix H.

Appendix M

Area Query MongoDB

//Code sourced from Darel Rex Finley (2011) Algorithm to find the area of a polygon [Source //code] https://www.mathopenref.com/coordpolygonarea2.html

```
db.getCollection('I7Polyedral').aggregate([
{$project:
{
Area: {$function: {
body: "function (geojson){
var xCoor = geojson.x; //loop through coordinate arrays, creating variables
var yCoor = geojson.y;
var coordArray = geojson.x;
var  numPoints= coordArray.length; // create a list of coordinates
var X= []; var Y =[];
 for (let i = 0; i < coordArray.length; i++) {(
X.push(xCoor[i]),
Y.push(yCoor[i]) ) }
  var area = 0;
var j; j = numPoints-1; // add the sum of cross multiplying the vertices offset by one then divide by two,e.g.,(x1*y2)
for ( var k =0; k<numPoints; k++) {
area +=  (X[j]+X[k]) * (Y[j]-Y[k]); j = k; }
return (Math.abs(area/2)); } ",
args: ["$json_geometry"],
lang: 'js'}}
}
}]).explain("executionStats")
```

Appendix M Area Query MongoDB. Syntax explanation, Appendix H.

Appendix N

Area Query PostGIS/PostgreSQL (SQL)

```
SELECT ST_Area(geom) FROM public."I1Polyedral";
```

Appendix N Area Query PostGIS/PostgreSQL (SQL). Syntax explanation, Appendix H.

Appendix O

Point in Polygon Query MongoDB

//Code sourced from :Rosetta Code, 2010. *Ray-casting algorithm.* [Online] Available at: https://rosettacode.org/wiki/Ray-casting_algorithm [Accessed 20 July 2020]

```
db.getCollection('A5Points').aggregate([{$project:
{ PIP: {$function: {
body: "function (bounds, north, east)
{var north = north[0];
var east = east[0];
var count = 0;
for (var b = 0; b < bounds.length; b++) {
var vertex1 = bounds[b];
 var vertex2 = bounds[(b + 1) % bounds.length];
 if (west(vertex1, vertex2, east, north)) ++count; } return count % 2;
function west(A, B, x, y) {
if (A.y <= B.y)
{if (y <= A.y || y > B.y || x >= A.x && x >= B.x)
{return false; }
else if (x < A.x && x < B.x)
{return true; .0 }
else {return (y - A.y) / (x - A.x) > (B.y - A.y) / (B.x - A.x); } }
else {return west(B, A, x, y); } } }",
args: [ [{x: 530186.6510982716, y:  185161.1715020783 },
        {x:  549053.7108801971, y: 185161.1631545085 },
        {x: 549053.7080037947, y:  194811.5583172982},
        {x: 530186.6357099144, y: 194811.5669819436},
        {x:530186.6510982716, y:  185161.1715020783 }],
"$json_geometry.y","$json_geometry.x"],
lang: 'js'}
}
}
}, {$match: {
PIP : { $gt: 0}
}}, {$group: {
_id: null,
count: {
$sum: 1
}
}}]).explain("executionStats")
```

Appendix O Point in Polygon Query MongoDB (Within). Syntax explanation, Appendix H.

Appendix P

2D & 3D Length MongoDB

// Code is based on Pythagorean theorem

```
db.getCollection('A5Points').aggregate(
[
{$project: {
"Length": {$function: {
body: "
function  (geojson) {
var coordArray = geojson.geometry.coordinates[0];
var  numPoints= coordArray.length;
var length =0;
for (let i = 0; i < coordArray.length-1; i++) {
length += Math.sqrt((
Math.pow((coordArray[i+1][0]-coordArray[i][0]), 2)+
Math.pow((coordArray[i+1][1]-coordArray[i][1]), 2)+
Math.pow((coordArray[i+1][2]-coordArray[i][2]), 2)
}
}
",args: ["$json_geometry.x"],
lang: 'js'}}
}
}
]
).explain("executionStats")
```

Appendix P 2D & 3D Length MongoDB. Syntax explanation, Appendix H

Appendix Q

2D Distance MongoDB Query

```
// Code is based on Pythagorean theorem

db.getCollection('A5Points').aggregate([{$project: {
"2D Distance": {$function: {
body: "function  (x1, y1,x2, y2) {
var x1 =  x1[0];var y1 =  y1[0];
 return Math.sqrt((Math.pow((x2-x1), 2)+ Math.pow((y2-y1), 2)) ) } "
,
args: ["$json_geometry.x","$json_geometry.y",534536.1399998665,185870.3899998665],
lang: 'js'}}
}}]).explain("executionStats")
```

Appendix Q 2D Distance Query MongoDB Syntax explanation, Appendix H

Appendix R

3D Distance MongoDB Query

```
db.getCollection('A5Points').aggregate([{$project: {
"3D Distance": {$function: {
body: "function  (x1, y1,z1,x2, y2,z2) {
var x1 =  x1[0];
var y1 =  y1[0];
 var z1 =  z1[0];
 return Math.sqrt((Math.pow((x2-x1), 2)+ Math.pow((y2-y1), 2)+ Math.pow((z2-z1), 2)) ) } "
,
args: ["$json_geometry.x","$json_geometry.y","$json_geometry.z",
534536.1399998665,185870.3899998665,11.1],
lang: 'js'}}
}}]).explain("executionStats")
```

Appendix R 3D Distance MongoDB Query. Syntax explanation, Appendix H

Appendix S

3D Distance PostGIS/PostgreSQL (SQL)

```
SELECT ST_3DDistance(geom,'SRID=27700 ;POINT(50 18 3)'::geometry) FROM
public."H1PointsDataset3D_Shapefile";
```

Appendix S 3D Distance PostGIS/PostgreSQL (SQL). Syntax explanation, Appendix H