



ROI Specification [DRAFT]

Release 0.0.1

Roger Leigh

October 23, 2012

CONTENTS

1	Introduction	1
1.1	Purpose	1
1.2	Scope	1
1.3	Reference implementation	1
1.4	Concepts	1
2	Requirements	3
2.1	Barcelona meeting	3
3	ROI Discussion	5
3.1	ROIs in three dimensions	5
3.2	Bitmasks	5
3.3	Meshes	5
3.4	Paths	5
4	2D primitives in 3D	7
4.1	Conversion to 3D primitives	7
4.2	Use of 2D primitives in 3D space	7
5	Fundamental data types	9
5.1	Basic primitives	9
5.2	Basic operators	10
5.3	Binary bitwise operators	10
6	Geometric shape primitives	11
6.1	Overview	11
6.2	Alternative shape representations	11
6.3	Shape serialisation	12
6.4	Shape	12
6.5	Polyline Splines	15
6.6	Polygon splines	15
6.7	Cylinders	16
6.8	Arcs	18
6.9	Masks	19
6.10	Meshes	21
6.11	Labels	22
6.12	Text placement and alignment	22
6.13	Scale bars	22

6.14	Additional primitives	23
6.15	Shape combination	23
7	Regions in arbitrary dimensions	25
7.1	Regions within a single dimension	25
7.2	Regions in multiple dimensions	26
8	Compound ROIs	27
8.1	Set primitives	27
8.2	Storing and manipulating complex compound objects	29
9	Compound types	33
9.1	Zeiss AxioVision ROI types	33
10	Affine transforms	35
10.1	2D transforms	35
10.2	3D transforms	35
11	Shapes	37
11.1	Overview	37
11.2	Definitions	38
11.3	Relationships	51
12	Shape representations	53
12.1	Overview	53
12.2	Definitions	56
13	Layers	73
14	ROI-ROI links	75
15	Storage of vertex data	77
15.1	XML schema	77
15.2	Properties	78
16	Definition of terms	79
17	Indices and tables	81
	Index	83

INTRODUCTION

1.1 Purpose

This document is a formal specification for the definition, storage and exchange of regions of interest (ROIs). This specification will be implementable in any programming language, and is intended to provide a common set of ROI types which will be usable in all image analysis software programs.

1.2 Scope

This specification defines abstract definitions of regions of interest, including details of how certain data structures and algorithms must be defined and behave, in order to ensure that ROIs work uniformly between the different programs and libraries implementing the specification. It also provides examples of serialised forms which may be used for storage and/or exchange. However, it does not define a file format; it is the responsibility of the implementors to integrate this model into their file formats as they see fit.

1.3 Reference implementation

This specification is accompanied by a reference implementation of the model. This implementation is intended to validate and test the correctness of the specification. It may be usable directly, however this is not the primary intention for its existence. Note that the reference implementation strives for complete correctness, and implementors of this specification may wish to provide additional optimisations to improve performance.

1.4 Concepts

- A ROI is an evaluation of a shape object
- A shape is defined by the rules which transform its representation (e.g. geometry, range within a dimension) into a a bitmask and/or greymask
- Each shape has a unique name (type) and number; the number is used for serialisation and versioning
- Each shape is described by one (or more) representations, these are the primitives which define the geometry or range within a dimension

- A shape object can be composed of one or more shapes, which can include transforms and shapes in arbitrary dimensions
- Each representation has a unique name and number; the number is used for serialisation and versioning
- Shapes which share representations may be freely interconverted; conversion is not required to be possible in both directions (e.g. square to rectangle or polyline to/from polygon)
- A shape is essentially a serialised expression which must be evaluated to create a usable ROI; given that certain shapes can contain other shapes, this provides for ROIs which are both extensible and of arbitrary complexity.
- All shapes can be serialised as a sequence of numbers
- Given that each shape can be reconstructed using its shape and representation numbers, which specify the exact sequence of numbers to deserialise to reconstruct the object, it is possible to exchange ROIs as simple text, or alternately as binary; more structured (but space inefficient) representations could be realised using XML.
- The object/function serialisation methodology used here is inspired by (but not derived from) the [SSH FXP specification](#) which defines the wire protocol for SFTP.

REQUIREMENTS

2.1 Barcelona meeting

The following points are taken from the meeting notes.

Note: These may be incomplete, please do correct if necessary.

- Iterate through all points in a ROI. Order is important for some use cases, but not all.
- Type mechanism needed. Use concrete types.
- Version types to allow for deprecation, and translation to new versions.
- Define a persistent data model.
- ROIs must be serialisable. Needed for exchange and persistence.
- Serialisation may be implementation dependent. This could be XML, text or binary. Which should be used as a transfer format (if any)?
- Allow conversion between different ROI types.
- Need the ability to specify an interval in an arbitrary dimension.
- **Hierarchy of interfaces.**
 - Interval, Renderable.
- Need to specify how to draw and display (and edit?) types such as curves so that it does not vary between implementations.
- Persistence and drawing are separate problems.
- **Transforms**
 - **Non-affine transforms.**
 - * Need examples to understand the problem better.
 - Store transforms with ROI?
- Needs to be able to scale up to millions-billions of ROIs
- **Specific ROI types**
 - Include checkerboard (uneven integers)

ROI DISCUSSION

3.1 ROIs in three dimensions

Some of the 3D primitives described below may appear to be redundant; it's certainly possible for example, to represent a shape in 3D right now using multiple shapes, one per z plane. However, being able to use native 3D primitives is more powerful: it permits additional measurements involving volume, surface area and shape.

Some shapes may be represented equivalently in different ways; it might be worth considering adding support for these because it firstly allows the shape to be computed in different ways, which can differ depending upon the problem being solved, and it also contains information about how the measurement was made, i.e. the intent of the person doing the measuring, which is lost if converted to a canonical form.

3.2 Bitmasks

Mask Could we have a pointer to an IFD/file reference plus two coordinates so specify a clip region, then we can pack potentially hundreds of masks in a single plane.

3.3 Meshes

Mesh 2D mesh described as e.g a face-vertex mesh.

3DMesh 3D mesh described as e.g. a face-vertex mesh.

Meshes could be computed from masks, polygons, extruded shapes where there is a z range, or from thresholding.

3.4 Paths

3DPath As for Path, but with additional vector to describe motion along the prescribed plane?

2D PRIMITIVES IN 3D

4.1 Conversion to 3D primitives

The existing 2D primitives may be represented by the equivalent 3D primitive for the 2D primitive, extruded in z to a single z plane thickness.

While this is desirable for reducing code complexity, retaining the 2D primitives is necessary for 2D measurements (area/perimeter). These can be obtained from the 3D shape by dividing the volume or surface area by the z thickness, respectively. Having the 2D primitives will provide the context for conversion of measurements from 3D volume to 2D surface, since these are otherwise meaningless for 3D ROIs which are not extruded 2D ROIs.

3D ROIs, where appropriate, could provide alternative forms for 2D use. For example, a 3D cylinder would, when extruded from a 2D circle, not have end faces (i.e. would be open), in order for 2D surface area measurements to be correct.

4.2 Use of 2D primitives in 3D space

While it would be possible to translate and rotate 2D primitives in 3D using a 4×4 matrix, it would be simpler for users if rotation could be specified using a unit vector which can specify the angle of the primitive in 3D space; the matrix transform can be trivially constructed from the vector. However, note that while current transforms occur only in 2D, where the x and y pixel sizes are typically the same, this is not usually the case in z, and so the transformations may need performing in physical units; therefore adding proper support for units would also be desirable to fully support 3D transforms. Note that this would also solve the existing problem in 2D that prevents ellipses and rectangles being rotated (without the use of a matrix transform), though where the rotation centre should be may be shape- and context-dependent. The unit vector to (0,0,-1) which would specify the existing behaviour.

Note: Define behaviour of orientation of unit vector for rotation; which direction are primitives facing by default?

4.2.1 2D extrusion

Reconstruction of 3D shapes from 2D planes distributed in z/t. -> set of 3D objects in t.

4.2.2 2D decomposition

Decompose 3D shape into 3D planes distributed in z.

FUNDAMENTAL DATA TYPES

The following defined types are used in the subsequent sections. Implementors should treat these sizes as minimum requirements.

Note: Roger Leigh Depending upon how we wish to pursue interoperability between implementations, these may be required to be exact. Using plain text would mitigate this to an extent.

5.1 Basic primitives

Table 5.1: Primitives

Primitive	Type	Description
ShapeID	uint16	Numeric shape identifier
RepID	uint16	Numeric shape representation identifier
Count	uint32	Number of objects
Index	uint32	Index into an array
Value	double	Numerical value
Operator	uint8	Mathematical operator
BLogic	uint8	Bitwise binary logical operator
Vertex1D	double[1]	Vertex in 1D
Vertex2D	double[2]	Vertex in 2D
Vertex3D	double[3]	Vertex in 3D
Vector1D	double[1]	Vector in 1D
Vector2D	double[2]	Vector in 2D
Vector3D	double[3]	Vector in 3D
Affine1D	double[2][2]	Affine transform in 1D
Affine2D	double[3][3]	Affine transform in 2D
Affine3D	double[4][4]	Affine transform in 3D

Table 5.2: Compound primitives

Primitive	SeqNo	Type	Name	Description
String	0	Count	NCHAR	Number of octets
String	1	uint8[NCHAR]	CHARS	Array of octets (UTF-8)

Note: Barry DeZonia Support different coordinate spaces as needed (int, long, double). Should be possible to iterate some regions.

5.2 Basic operators

Operator	Value	Description
=	0	Equals
≠	1	Not equals
<	2	Less than
≤	3	Less than or equal to
>	4	Greater than
≥	5	Greater than or equal to

5.3 Binary bitwise operators

Operator	Value	Description
	0	And
	1	Or
¬	2	Not
	3	Exclusive or

All shape primitives are described in terms of the above fundamental primitives. This means that all shape descriptions are serialisable as a list of integer and double-precision floating point values. The specifics of this are implementation-defined. Example formats:

- Plain text, as a list of values
- XML, as element content or a string attribute
- Binary data stream, using big-endian/network byte order

This also means that for compatible shape types, the shape type may be changed while retaining the following data unchanged (e.g. polyline to polygon spline with the same point list).

Note: Roger Leigh All 2D shape primitives could be oriented in 3D or using a unit Vector3D, which would allow all 2D shapes to be used as surfaces in 3D. They would additionally require a depth in order to be meaningful (or assume a depth of one z slice).

Or, 2D shapes should specify the pair of x/y/z axes they are using, and will be extruded along the third axis.

GEOMETRIC SHAPE PRIMITIVES

6.1 Overview

This section specifies how shapes are described in the model. For some shapes, there are several alternative ways of specifying them; which are worth supporting needs further discussion. One point to consider is that the different ways preserve the intent behind the original measurement and what is in the original metadata where this makes sense, even if this does mean some redundancy; this won't impact on the actual drawing/analysis code, which can deal with each shape in a canonical form. This records how the measurement was made by the user, which may have implications in further analysis and/or verification that the measurement was correct.

While some shapes have been included here for completeness, it's quite possible that not all are needed, particularly in all dimensions.

If anyone wants to check the maths behind the geometry, that would be much appreciated, because I'm firstly not an expert in this area, and it's also quite possible I've made some typos. The naming of the shapes is probably also wanting some improvement.

6.2 Alternative shape representations

Using the current ROI model is that there is only one way to describe each shape. e.g. a polyline can only be described as a series of points; it might in some cases be more natural to specify one as a starting point and a series of vectors; while either are fine just to draw the ROI, it would be desirable to store what was measured, since converting it to a canonical representation is lossy, and removes the original measurements taken, and hence the intent of the original annotation. This applies to other shapes as well. For example, a circle or ellipse can be described by a bounding box (which may itself be a point and one or two vectors, or a set of points), or by a point and radius or half-axes, or by the Mahalanobis distance (typically for computing from a normal distribution of points). For a cylinder/cone, we can specify this in multiple ways also from a circle/ellipse plus length, or point plus vector (length and direction) plus radius (or half-axes).

The current model is focussed on drawing shapes, while making measurements involves drawing only for visualisation; the important parts are the values for making the measurement, and of course the results. Some programs (e.g. AxioVision) have separate sets of objects for drawing (annotation) and measurement. These are a largely overlapping set, but the former are not used for any length/area/volume/pixel measurements. Objects such as scale bars and labels are for drawing only.

Common methods for all primitives: Bounding box [AlignedCuboid3D] Rotation centre [Vertex2D/Vertex3D] Control points [may use points and vertex to describe position and movement path] Conversion to 2D (slab through); equivalent to intersection with cuboid. Should all primitives support a minimum of intersection with AlignedCuboid3D? Or Mesh3D for non-square images. Can 2D methods use alternative axes to project in xz/yz? Default to xy. If all 2D shapes must be represented by 3D forms (i.e. are just proxies), then the equivalent 3D can be used quite simply. Get greymap/bitmap. Get 2D/3D mesh. Intersect (only for cuboid?) Need to clip to image volume (optionally). Also useful to reduce to 2D (which can be a cuboid for a single plane). Non-aligned shapes inherit/implement the aligned forms. Shrink and grow: move polygons along surface normals for meshes. For other shapes, this will require recalculation of the geometry.

Add triangle as special case of polygon, which can be a special case of mesh?

Meshes: Need to be able to triangulate if higher order polygons are possible.

Add representation number to start of number list; this will allow shapes to be embedded in other shapes and be self-describing. e.g. all circle types may be used to specify a circular cylinder end. This will simplify the specification of more complex shapes by limiting the number of variants.

6.3 Shape serialisation

Key considerations:

- A shape exists in a set of dimensions e.g. xy, xyz, xyt. The shape must define the number of dimensions it exists in, and their identity.
- A shape must be identifiable unambiguously
- A shape must be versioned (to permit correction of any design/analysis bugs without altering any data retrospectively); this permits the replacement of the buggy implementation while not removing it.
- In order to allow code reuse and flexible use of shapes, shapes may include other shapes as part of their primitive specification.

In the following shape descriptions, all shapes are identified by a Shape ID and Representation ID. The shape specifies the geometric shape type. The representation specifies both the primitives required for serialisation, and can also be used for versioning the shape—i.e. it also specifies the behaviour for conversion to greymaps and bitmaps.

6.4 Shape

An abstract description of a shape.

Representation:

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation

Concrete implementations of shapes provide further elements in their representation. The above are only sufficient to describe the shape and its representation. The combination of shape and representation specifies the data required to construct the shape.

Note that one disadvantage of this method is that a reader will be required to understand how to deserialise all shape types; it's not possible to skip unknown shapes due to not knowing their lengths (which may be variable). However, this would be an issue for a purely XML-based implementation as well, so may not be a problem in practice.

When a shape embeds a specific shape, it may skip the ShapeID and/or RepID header if one or both of these are fixed. If both headers are skipped, this is indicated with a '*', or if only the ShapeID header is skipped, this is indicated with '&':

Type	Description
Cube3D	Shape contains a 3D cube
Cube3D*	Shape contains a 3D cube with no header
Cube3D&	Shape contains a 3D cube with RepID only

Alignment Aligned shape variants are aligned at right-angles to the x and y (2D) or x, y and z (3D) axes.

6.4.1 Circle2D

Todo

Specify using bounding square. Inherits all Square2D and AlignedSquare2D representations.

Specify using diameter (two points)

6.4.2 Sphere3D

Todo

Specify using bounding cube . Inherits all Cube3D and AlignedCube3D representations. Specify using diameter (two points)

Todo

Specify using reversed radius (vector3D to centre) Specify using diameter (two points) Specify using 4 points around surface (->radius and centre)

6.4.3 AlignedEllipse2D

Representation 2: Bounding rectangle. Inherits all AlignedRectangle2D representations.

6.4.4 Ellipse2D

Representation 2: Bounding rectangle: Inherits all Rectangle2D and AlignedRectangle2D representations.

Representation 3: Mahalanbobis distance used to draw an ellipse using the mean coordinates (P1) and 2×2 covariance matrix (COV1)

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex2D	Centre point (mean)
COV1	double[4]	2×2 covariance matrix

6.4.5 AlignedEllipsoid3D

Aligned at right angles to xyz axes.

Representation 1: Centre and half axes

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex3D	Centre point
V1	Vector3D	Half axes (x,y,z)

Representation 2: Centre and half axes (specified separately).

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex3D	Centre point
V1	Vector3D	Half axis (x)
V2	Vector3D	Half axis (y)
V3	Vector3D	Half axis (z)

Representation 3: Bounding cuboid: Inherits all AlignedCuboid3D representations.

6.4.6 Ellipsoid3D

May be rotated; not aligned at right angles to xyz axes.

Representation 1: Centre and half axes; V2 and V3 are at right-angles to V1 and each other, so have reduced dimensions.

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex3D	Centre point
V1	Vector3D	Half axes (xyz)
V2	Vector2D	Half axes (xy)
V3	Vector1D	Half axes (x)

Representation 2: Bounding cuboid: Inherits all Cuboid3D and AlignedCuboid3D representations.

Representation 3: Mahalanbobis distance used to draw an ellipse using the mean coordinates (P1) and 3×3 covariance matrix (COV1)

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex3D	Centre point (mean)
COV1	double[9]	3×3 covariance matrix

6.5 Polyline Splines

6.5.1 PolylineSpline2D

Representation:

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
NPOINTS	Count	Number of points
P1	Vertex2D	Line start
...	Vertex2D	Further points
Pn	Vertex2D	Line end

6.5.2 PolylineSpline3D

Representation:

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
NPOINTS	Count	Number of points
P1	Vertex3D	Line start
...	Vertex3D	Further points
Pn	Vertex3D	Line end

6.6 Polygon splines

6.6.1 PolygonSpline2D

Representation:

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
NPOINTS	Count	Number of points
P1	Vertex2D	Line start
...	Vertex2D	Further points
Pn	Vertex2D	Line end

6.6.2 PolygonSpline3D

Representation:

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
NPOINTS	Count	Number of points
P1	Vertex3D	Line start
...	Vertex3D	Further points
Pn	Vertex3D	Line end

6.7 Cylinders

6.7.1 AlignedCircularCylinder3D

Aligned

6.7.2 CircularCylinder3D

Representation 1: Start and endpoint, plus radius.

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex3D	Centre of first face
P2	Vertex3D	Centre of second face
V1	Vector1D	Radius

Representation 2: Start point, distance to endpoint, plus radius

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex3D	Centre of first face
V1	Vector3D	Distance to centre of second face
V2	Vector1D	Radius

Representation 3: Start and endpoint, plus vectors to define radius (V1) and angle of start face, and unit vector defining angle of end face. Face angles other than right-angles let chains of cylinders be used for tubular structures without gaps at the joins.

Note: Should V2 only allow angle, assuming radius from V1, or also allow a second radius to represent a conical section?

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex3D	Centre of first face
P2	Vertex3D	Centre of second face
V1	Vector3D	Radius and angle of first face
V2	Vector3D	Angle of second face

Representation 4: Start point, distance to endpoint, plus vectors to define radius (V2) and angle of start face, and unit vector defining angle of end face (V3). Face angles other than right-angles let chains of cylinders be used for tubular structures without gaps at the joins.

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex3D	Centre of first face
V1	Vector3D	Distance to centre of second face
V2	Vector3D	Radius and angle of first face
V3	Vector3D	Angle of second face

Note: Should V3 only allow angle, assuming radius from V2, or also allow a second radius to represent a conical section?

6.7.3 AlignedEllipticCylinder3D

Todo

Inherits from AlignedEllipse.

6.7.4 EllipticCylinder3D

Representations 1 and 2 describe basic elliptic cylinders with faces at right angles; the following representations permit faces at arbitrary angles. Face angles other than right-angles let chains of cylinders be used for tubular structures without gaps at the joins.

Representation 1: Start and endpoint, plus half axes.

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex3D	Centre of first face
P2	Vertex3D	Centre of second face
V1	Vector2D	Half axes (xy)
V2	Vector1D	Half axes (x)

Note: Is the dimensionality of the half axes correct here?

Representation 2: Start point, distance to endpoint, plus half axes

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex3D	Centre of first face
V1	Vector3D	Distance to second face
V2	Vector3D	Half axes (xy)
V3	Vector2D	Half axes (x)

Note: Is the dimensionality of the half axes correct here?

Todo

Should half axes and angle be specified in same vector or separately?

3: Start and endpoint, plus vectors to define half axes (V1 and V2) and angle of start face, and unit vector defining angle of end face (V3).

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex3D	Centre of first face
P2	Vertex3D	Centre of second face
V1	Vector3D	Half axes of first face (xyz)
V2	Vector2D	Half axes of first face (xy)
V3	Vector3D	Angle of second face

3: Start and endpoint, plus vectors to define half axes (V1 and V2) and angle of start face, and unit vector defining angle of end face (V3).

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex3D	Centre of first face
V1	Vector3D	Distance to second face
V2	Vector3D	Half axes (xyz)
V3	Vector2D	Half axes (xy)
V4	Vector3D	Angle of second face

Representation 4: Bounding cuboid: Inherits all Cube3D and Cuboid3D representations; first face is the base.

6.8 Arcs

6.8.1 Arc2D

Representation 1:

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex2D	Centre point
P2	Vertex2D	Arc start
V1	Vector2D	Arc end

Representation 2:

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex2D	Centre point
V2	Vector2D	Arc start
V1	Vector2D	Arc end

6.8.2 Arc3D

Representation 1:

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex3D	Centre point
P2	Vertex3D	Arc start
V1	Vector3D	Arc end

Representation 2:

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex3D	Centre point
V2	Vector3D	Arc start
V1	Vector3D	Arc end

6.9 Masks

Masks may be either grey masks (double or integer) or bitmasks.

For all of the following masks, DATA should be stored outside the ROI specification either as BinData or (better) in an IFD for OME-TIFF. It could be stored as part of the double array, but this would be quite inefficient.

Note: Masks are applied to the bounding rectangle, and so a 1:1 correspondance between mask and image pixel data is not required. In this case, a new greymask should be computed which is aligned with the pixel data, and then (if required) thresholded to a bitmask.

6.9.1 GreyMask2D

Representation:

The mask is applied to the bounding rectangle. Dimensions specify the x and y size of the mask. DATA is the mask pixel data.

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex2D	Start point of bounding rectangle
P2	Vertex2D	End point of bounding rectangle
DIM1	Vector2D	Mask dimensions (x,y)
DATA	double[x,y]	Mask data

6.9.2 BitMask2D

Representation:

The mask is applied to the bounding rectangle. Dimensions specify the x and y size of the mask. DATA is the mask pixel data.

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex2D	Start point of bounding rectangle
P2	Vertex2D	End point of bounding rectangle
DIM1	Vector2D	Mask dimensions (x,y)
DATA	bool[x,y]	Mask data

6.9.3 GreyMask3D

Representation:

The mask is applied to the bounding cuboid. Dimensions specify the x, y and z size of the mask. DATA is the mask pixel data.

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex3D	Start point of bounding rectangle
P2	Vertex3D	End point of bounding rectangle
DIM1	Vector3D	Mask dimensions (x,y)
DATA	double[x,y,z]	Mask data

6.9.4 BitMask3D

Representation:

The mask is applied to the bounding cuboid. Dimensions specify the x, y and z size of the mask. DATA is the mask pixel data.

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex3D	Start point of bounding rectangle
P2	Vertex3D	End point of bounding rectangle
DIM1	Vector3D	Mask dimensions (x,y)
DATA	bool[x,y,z]	Mask data

6.10 Meshes

Mesh representation depends upon the mesh format. In the examples below, face-vertex meshes are used.

6.10.1 Mesh2D

Representation:

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
NFACE	Count	Number of faces
VREF	double[NFACE][3]	Vertex references per face, counterclockwise winding
NVERT	Count	Number of vertices
VERTS	Vertex2D[NVERT]	Vertex coordinates

Vertex references are indexes into the VERTS array. Vertex-face mapping is implied, and will require the implementor to construct the mapping.

6.10.2 Mesh3D

Representation:

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
NFACE	Count	Number of faces
VREF	double[NFACE][3]	Vertex references per face, counterclockwise winding
NVERT	Count	Number of vertices
VERTS	Vertex3D[NVERT]	Vertex coordinates

Vertex references are indexes into the VERTS array. Vertex-face mapping is implied, and will require the implementor to construct the mapping.

6.11 Labels

6.12 Text placement and alignment

In order to annotate text next to measurements, it would be ideal if it were possible to control text placement and orientation. Currently the coordinate of the first letter is required. However, it would be nicer if the text could be also placed to the right of the point or centred on the point. And additionally, to the top, middle or bottom for vertical placement. Rotation would also be useful, though it's probably achievable indirectly via the transformation matrix, i.e. you would effectively have these anchors for placement, where 1 is the current behaviour.

7	8	9
4Text	h5ere...	6
1	2	3

This is needed to e.g. align text along measurement lines. Having a rotation angle specified directly would also save the need for complex calculations to work out the rotation origin and transform every time you want to just place a label along a line. It also makes it possible to place text in the centre of a shape.

6.12.1 Text2D

Representation 1: Text aligned relative to a point. Inherits all Point2D and Point3D representations.

Representation 2: Text aligned relative to a line. Inherits all Line2D and Line3D, Direction2D and Direction3D representations.

Representation 3: Text aligned and flowed inside a rectangle. Inherits all AlignedSquare2D, Square2D, AlignedRectangle2D and Rectangle2D representations.

6.13 Scale bars

6.13.1 Scale2D

Representation 1: Scale bar between two points. Inherits all Line2D representations.

Representation 1: Scale bar described by vector. Inherits all Distance2D representations.

6.13.2 Scale3D

Representation 1: Scale bar between two points. Inherits all Line3D representations

Representation 1: Scale bar described by vector. Inherits all Distance3D representations.

Note: A 3D scale may need to be a 3D grid to allow visualisation of perspective, in which case the representation will define the grid bounding cuboid; inherit AlignedCuboid3D representations. Permit scale rotation with Cuboid3D? Allow specification of grid size and only allow sizing in discrete units?

6.14 Additional primitives

3D spline surfaces Natural cubic spline (Catmull-Rom)

The axiovision curve type is most likely a natural cubic spline, the curve passing smoothly through all points, but without local control. It is simply represented as a list of points through which the curve must pass; there are no additional control points. Depending upon if they are doing any custom stuff, it might not be possible to represent with pixel-perfect accuracy.

Curves might be more generally applicable to other formats, and useful in their own right. It might be worth considering adding a spline type with local control where the curve passes straight through the control points such as Catmull-Rom splines. This would make it very simple for non-experts to fit smooth lines while annotating their images.

6.15 Shape combination

6.15.1 Logical combination

Representation:

Logical combination of two shapes using the formula $S2 \ B1 \ S3$, e.g $S2 \ S3$. Shapes must have compatible dimensions.

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
B1	BLogic	Bitwise binary logical operator
S2	ShapeID	Shape1
S3	ShapeID	Shape2

6.15.2 Concatenation

Logical combination of shapes. Shapes must not share any dimensions, i.e. this is a means of adding additional dimensions to shapes. The purpose is to enable combination of 2D or 3D geometry with additional nD shapes.

Representation:

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
NSHAPES	Count	Number of shapes
SHAPES	ShapeID[NSHAPES]	Shapes to combine

REGIONS IN ARBITRARY DIMENSIONS

While it is possible to use geometric shapes to specify regions in physical dimensions, this does not translate meaningfully to arbitrary dimensions. The following primitives work in any “dimension” with a discrete or continuous range by permitting the selection of specific values, or sub-ranges.

These representations could be represented efficiently using a 1D bitmap or Line1D.

7.1 Regions within a single dimension

7.1.1 ValuenD

Constrain region to a single value within a specific dimension.

Representation:

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
D1	Index	Dimension
V1	Value	Value within dimension

7.1.2 ValuesnD

Constrain region to multiple values within a specific dimension.

Representation:

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
D1	Index	Dimension
NVAL	Count	Number of values
VALS	Value[NVAL]	Values within dimension

7.1.3 RangenD

Constrain region to a range of values within a specific dimension.

Representation1:

Specified as the half-open range [V1, V2).

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
D1	Index	Dimension
V1	Value	Starting value within dimension
V2	Value	Ending value +1 within dimension

Representation2:

As an inequality (or equality).

Specified as all values for which the formula “n O1 V1” is true, e.g. “ $n \leq 5$ ”.

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
D1	Index	Dimension
O1	Operator	Mathematical operator
V1	Value	Value for operation

7.2 Regions in multiple dimensions

Regions in single dimensions may be combined to result in a region in multiple dimensions.

7.2.1 ExtrudenD

Extrude a shape of arbitrary dimensionality into an additional dimension. There are no limits in the additional dimension; these must be set by combining with a range instead.

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
D1	Index	Dimension
S2	ShapeID	Shape to extrude

7.2.2 CombinenD

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
S2	ShapeID	First shape to combine
S3	ShapeID	Second shape to combine

COMPOUND ROIS

A ROI may consist of multiple shapes combined in different ways. The result is also a shape.

8.1 Set primitives

Shapes may combined using set operators:

- union
- intersection
- difference
- symmetric difference

The shape is the result of the set operation.

Note: Restrict to combinations of 2D or 3D shapes only?

Note: **J-M Burel** Union in the mathematical sense or aggregation.

8.1.1 Set

A simple collection of shapes. There is no implied relationship unless used with the set operators.

Representation:

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
NSHAPE	Count	Number of shapes
SHAPE1	Shape	First shape
...	Shape	Further shapes
SHAPE _n	Shape	Last shape

8.1.2 Union

Produce the union of the shapes in the provided set.

Representation:

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
SET	Set*	Set of shapes

8.1.3 Intersection

Produce the intersection of the shapes in the provided set.

Representation:

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
SET	Set*	Set of shapes

8.1.4 Difference

Produce the set difference of the shapes in the provided set.

Representation:

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
SET	Set*	Set of shapes

8.1.5 Symmetric difference

Produce the symmetric difference of the shapes in the provided set.

Representation:

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
SET	Set*	Set of shapes

- Restrict to either 2D or 3D, but not both?

How do we detect if shapes intersect? Edge cases for set operations using masks-false positives for partially occupied pixels.

Event/Events: A simple list of points. The point size/style/colour may be changed to permit different sets to be distinguished.

Caliper/Distance/Multi-Caliper/Multi-Distance. These are all the same measurement(s), a baseline followed by a list of points. The measurement is the distance from each point to the baseline. The differences between the types are solely the visual presentation of the measurements.

Angle3/Angle4. These measure the angle between two lines. **Angle4** is two separate lines, while **Angle3** is two lines with a common point (i.e. a special case of **Angle4**). **Angle3** could be represented with a three-point polyline. **Angle4** would need to be two separate lines. Given that **Angle3** is a special case of **Angle4**, it is not clear that it should be represented as a polyline.

Circle. While the OME model represents this as an ellipse with equal x and y radii, there are three ways to represent a circle here: - radius defined as a line from centre to edge - radius defined as a line from edge to centre (stored as the first type with the point order reversed) - circumference defined using three points. The first two are representable in the model as an ellipse plus a line. The latter is representable as an ellipse plus three points.

Polyline is directly translatable.

Aligned Rectangle is directly translatable as a rectangle (with some trivial differences in coordinates). However, additional tags define metadata to display inside the rectangle (optional) such as channel/slide/acquisition time/exposure time/etc. The verbatim text can be put into a Label, but the specific meaning would be lost—this is an overlay which would change as you navigate through a stack or timecourse etc, varying with the plane-specific parameters. While the specific tags would be retained, a more generic means to overlay image- and plane-specific OME metadata might be generally useful within the context of the OME model.

Ellipse is directly translatable.

Outline/closed polyline is directly translatable.

Text is convertible to Label. However, the OME Label type lacks the alignment attributes mentioned in my earlier mail. This makes it difficult to control the placement of text in complex compound ROIs.

Length is a single line distance measurement line like, but with additional end lines to make it like a technical drawing line outside the object itself, i.e.

```
| *****OBJECT***** |
|                         |
| <----->             |
|           50 µm        |
```

Representable in the model as a simple line, across OBJECT, but with loss of the other lines. It is representable as three separate lines, but with loss of the context of the specific measurement.

Open and closed splines: these are probably natural splines (not Bezier). ZVI currently stores them as polylines given that we don't support splines. But having a spline type would permit them to be stored.

LUT and Profile: Covered in previous mail.

8.2 Storing and manipulating complex compound objects

With these measurements, one thing perhaps worth considering is that there are up to four types of object here:

1. Result context: the object(s) representing the physical measurement. This is what we currently store in the model.
2. Measurement context: line along radius of circle, points along circumference of circle etc. This is “how the measurement was made”
3. Visual context: such as visual cues such as construction lines. This is the visual presentation of the measurement to the viewer.
4. Editing context: values which control the placement of the above. Information for generation of UI manipulation handles, and of the other contexts while editing.

We can represent the actual measurements in most cases using the existing ROI types. However, if we store the additional types, it is no longer possible to distinguish between the measurement and the additional context.

If it was possible to distinguish between these in the model, it would be possible for the objects to be displayed without any advanced knowledge of how an object should be edited. It would also be possible to extract the primitive measurement values. However, the measurement context would provide additional information to editors for manipulation of the object, which would then be able to update all three contexts appropriately.

Doing this would provide a simple but effective means for additional ROI types to be added without requiring support in all programs displaying/modifying ROIs. This does not of course replace the need for namespaces to identify ROI categories, but it does supplement it by allowing programs to selectively display different contexts without any knowledge of the underlying type.

As an example, using this length measurement:

```
| *****OBJECT***** |  
|                         |  
| <-----> |  
          50 μm
```

1. Result context

```
#*****OBJECT*****#
```

(where the #s are the start and end points of a Line at either end of the object. This is the value of the physical measurement.)

2. Measurement context

No additional information needed in this case.

3. Visual context

```
|                         |  
|                         |  
| <-----> |  
          50 μm
```

Three lines, one with arrow end markers, plus text label.
This is the visual representation of the measurement.

4. Editing context

```
*****OBJECT*****  
#  
#
```

(where the #s represent a distance between the measured line and the drawn line in the visual context. This information is used to generate the visual context from the measurement context.)

I hope the above does not sound too way out. But the current system is limited to storing only the first of these four contexts, which loses information. While it is possible to delegate all of the presentation and editing to the viewer, the reality is that this is stuff people want. If I'm annotating an image for a paper, I want the annotations to appear exactly the same as I see them if I send them to someone else. And if I'm doing physical measurements, I want the specifics of how I made the measurement to be recorded. All we are doing here is providing additional information to the viewer/editor that it is free to use and/or ignore as it chooses.

Thinking about this a little more, in many cases it will be possible to omit some contexts and infer them from the others. For example, if I have a simple line I will store a line in the result context. The measurement context is the same two points, and so we may simply use the result context points in its place. Likewise, if the measurement is a simple one, the visual context may be omitted and inferred from the result context also. The different contexts really only come into play when we want a more sophisticated visual representation (for example with overlaid textual representations of the measurement value or to visualise the measurement in a more complex manner than the result context alone can provide). And they are essential when using more complex compound ROIs as the last example attached shows.

In the last example, all the information is provided to allow the user to edit the object in a UI. For example, they can adjust the end points of the baseline, and the start points of the lines in the measurement context can be retriangulated from the end points and baseline. The measurement context can be inferred from the endpoints of the lines in the result context. And the endpoints can also be adjusted independently. Following any adjustment, the updated baseline can be stored in the editing context, the measurement lines in the measurement context, and the visual representation in the visual context. The visual context is shown here to include end markers on the distance lines, and text labels with the measured values. But these could be toggled on or off and the settings stored in an annotation specific for this measurement type—there's really no limit to the “extra stuff” you can add here, but the basic measurement remains the same in the result context.

(In this example, the baseline could actually be in the measurement context, since it's part of the measurement; the first example is a better illustration of the editing context.)

The important point is that anyone should be able to open the file and display the visual representation without any knowledge of the specifics of the ROI type or measurements being made. Likewise they can also look at the measured distances in the results context and use them without any knowledge of how they were measured. Only a UI which supports the ROI type in question will need to use the editing and/or measurements context, and they will know how to regenerate the other contexts when editing.

COMPOUND TYPES

Line Profile LUT Scale bar

LUT/gradient boxes are quite specialist. However, they are also quite common in published figures, so it would make sense to have a general implementation. These are particularly useful when you have false colour heat maps where you need a visual scale to interpret the figure. We already support LUTs, so this is really just a view of the LUT for a given channel inside a rectangle.

Line profiles are quite common. But I guess supporting this would depend upon whether you classify the profile as the result of analysis of a ROI, or part of a ROI. It might be handy to be able to overlay a line profile as a set of coloured polylines, for example.

9.1 Zeiss AxioVision ROI types

For the Zeiss types, we can represent these in the model using:

Zeiss type	ROI model type
Event	Point2D
Events	Point2D (union of points)
Line	Line2D
Caliper	Line2D (union of lines)
Multiple caliper	Line2D (union of lines)
Distance	Line2D (union of lines)
Multiple distance	Line2D (union of lines)
Angle3	Line2D and Arc2D
Angle4	Line2D and Arc2D
Circle	Circle2D and Line2D
Scale Bar	Line2D (with end markers)
Polyline [open]	Polyline2D
Aligned Rectangle	AlignedRectangle2D
Rotated Rectangle	Rectangle2D
Ellipse	AlignedEllipse2D
Polyline [closed]	Polygon2D
Text	Label2D
Length	Line2D (union of lines)
Spline [open]	PolylineSpline2D
Spline [closed]	PolygonSpline2D
LUT	AlignedRectangle2D and Label2D
Line profile	Line2D and Polyline2D/Rectangle2D

Annotations don't typically have labels (with the exception of scale bars). Measurements would have one or more labels in the union as well displaying the value(s) of the measurement.

AFFINE TRANSFORMS

To support proper 3D operation, it would make sense to extend the existing support for 3×3 2D affine transforms to 4×4 3D transforms.

For both 2D and 3D transforms, translation, rotation and scaling are supported. Skewing, using the bottom row of the matrix, is not.

10.1 2D transforms

$$\begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix}$$

10.2 3D transforms

$$\begin{bmatrix} a & d & g & j \\ b & e & h & k \\ c & f & i & l \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

SHAPES

11.1 Overview

Table 11.1: Shapes

ID	Shape	Dims	Description
0	<i>Point</i>	2D	A single point
1	<i>Point</i>	3D	A single point
2	<i>Points</i>	2D	A set of points
3	<i>Points</i>	3D	A set of points
4	<i>Line</i>	2D	A single line
5	<i>Line</i>	3D	A single line
6	<i>Lines</i>	2D	A set of lines
7	<i>Lines</i>	3D	A set of lines
8	<i>Polyline</i>	2D	A set of connected points
9	<i>Polyline</i>	3D	A set of connected points
10	<i>Polygon</i>	2D	A polygon (closed polyline)
11	<i>Polygon</i>	3D	A “polygon” (closed polyline) [does this make sense to have in 3D?]
14	<i>AlignedSquare</i>	2D	A square aligned with all axes
15	<i>Square</i>	2D	A square
16	<i>AlignedCube</i>	3D	A cube aligned with all axes
17	<i>Cube</i>	3D	A cube
18	<i>AlignedRectangle</i>	2D	A rectangle aligned with all axes
19	<i>Rectangle</i>	2D	A rectangle
20	<i>AlignedCuboid</i>	3D	A cuboid aligned with all axes
21	<i>Cuboid</i>	3D	A cuboid
22	<i>Circle</i>	2D	A circle
23	<i>Sphere</i>	3D	A sphere
24	<i>AlignedEllipse</i>	2D	An ellipse aligned with all axes
25	<i>Ellipse</i>	2D	An ellipse
26	<i>AlignedEllipsoid</i>	3D	An ellipsoid aligned with all axes
27	<i>Ellipsoid</i>	3D	An ellipsoid
28	<i>PolylineSpline</i>	2D	A set of connected splines (open)
29	<i>PolylineSpline</i>	3D	A set of connected splines (open)
Continued on next page			

Table 11.1 – continued from previous page

ID	Shape	Dims	Description
30	<i>PolygonSpline</i>	2D	A set of connected splines (closed)
31	<i>PolygonSpline</i>	3D	A set of connected splines (closed)
33	<i>CircularCylinder</i>	3D	A circular cylinder
35	<i>EllipticCylinder</i>	3D	An elliptic cylinder
36	<i>Arc</i>	2D	An arc
37	<i>Arc</i>	3D	An arc
38	<i>BitMask</i>	1D	A mask with one bit values
39	<i>BitMask</i>	2D	A mask with one bit values
40	<i>BitMask</i>	3D	A mask with one bit values
41	<i>GreyMask</i>	1D	A mask with multiple grey levels
42	<i>GreyMask</i>	2D	A mask with multiple grey levels
43	<i>GreyMask</i>	3D	A mask with multiple grey levels
44	<i>Mesh</i>	2D	A mesh
45	<i>Mesh</i>	3D	A mesh
46	<i>AffineTransform</i>	1D	Affine transformation of a shape
47	<i>AffineTransform</i>	2D	Affine transformation of a shape
48	<i>AffineTransform</i>	3D	Affine transformation of a shape

11.2 Definitions

11.2.1 Point (2D)

A single point.

Table 11.2: Point representations (2D)

Representation	Dim	In	Out	Inherited from
<i>RPoint</i>	2D	•	•	Point (2D) [self]

Canonical form is RPoint (2D).

11.2.2 Point (3D)

A single point.

Table 11.3: Point representations (3D)

Representation	Dim	In	Out	Inherited from
<i>RPoint</i>	3D	•	•	Point (3D) [self]

Canonical form is RPoint (3D).

11.2.3 Points (2D)

A set of points.

Table 11.4: Points representations (2D)

Representation	Dim	In	Out	Inherited from
<i>RPoints</i>	2D	•	•	Points (2D) [self]

Canonical form is RPoints (2D).

11.2.4 Points (3D)

A set of points.

Table 11.5: Points representations (3D)

Representation	Dim	In	Out	Inherited from
<i>RPoints</i>	3D	•	•	Points (3D) [self]

Canonical form is RPoints (3D).

11.2.5 Line (2D)

A single line.

Table 11.6: Line representations (2D)

Representation	Dim	In	Out	Inherited from
<i>RLinePoints</i>	2D	•	•	Line (2D) [self]
<i>RLineVector</i>	2D	•	•	Line (2D) [self]

Canonical form is RLinePoints (2D).

11.2.6 Line (3D)

A single line.

Table 11.7: Line representations (3D)

Representation	Dim	In	Out	Inherited from
<i>RLinePoints</i>	3D	•	•	Line (3D) [self]
<i>RLineVector</i>	3D	•	•	Line (3D) [self]

Canonical form is RLinePoints (3D).

11.2.7 Lines (2D)

A set of lines.

Table 11.8: Lines representations (2D)

Representation	Dim	In	Out	Inherited from
<i>RLinePoints</i>	2D			Line (2D)
<i>RLineVector</i>	2D			Line (2D)
<i>RLinesPoints</i>	2D	•	•	Lines (2D) [self]
<i>RLinesVectors</i>	2D	•	•	Lines (2D) [self]

Canonical form is *RLinesPoints* (2D).

11.2.8 Lines (3D)

A set of lines.

Table 11.9: Lines representations (3D)

Representation	Dim	In	Out	Inherited from
<i>RLinePoints</i>	3D			Line (3D)
<i>RLineVector</i>	3D			Line (3D)
<i>RLinesPoints</i>	3D	•	•	Lines (3D) [self]
<i>RLinesVectors</i>	3D	•	•	Lines (3D) [self]

Canonical form is *RLinesPoints* (3D).

11.2.9 Polyline (2D)

A set of connected points.

Table 11.10: Polyline representations (2D)

Representation	Dim	In	Out	Inherited from
<i>RPolylinePoints</i>	2D	•	•	Polyline (2D) [self]
<i>RPolylineVector</i>	2D	•	•	Polyline (2D) [self]

Canonical form is *RPolylinePoints* (2D).

11.2.10 Polyline (3D)

A set of connected points.

Table 11.11: Polyline representations (3D)

Representation	Dim	In	Out	Inherited from
<i>RPolylinePoints</i>	3D	•	•	Polyline (3D) [self]
<i>RPolylineVector</i>	3D	•	•	Polyline (3D) [self]

Canonical form is *RPolylinePoints* (3D).

11.2.11 Polygon (2D)

A polygon (closed polyline).

Table 11.12: Polygon representations (2D)

Representation	Dim	In	Out	Inherited from
<i>RPolylinePoints</i>	2D	•	•	Polygon (2D) [self]
<i>RPolylineVector</i>	2D	•	•	Polygon (2D) [self]

Canonical form is *RPolylinePoints* (2D).

11.2.12 Polygon (3D)

A “polygon” (closed polyline) [does this make sense to have in 3D?].

Table 11.13: Polygon representations (3D)

Representation	Dim	In	Out	Inherited from
<i>RPolylinePoints</i>	3D	•	•	Polygon (3D) [self]
<i>RPolylineVector</i>	3D	•	•	Polygon (3D) [self]

Canonical form is *RPolylinePoints* (3D).

11.2.13 AlignedSquare (2D)

A square aligned with all axes.

Table 11.14: AlignedSquare representations (2D)

Representation	Dim	In	Out	Inherited from
<i>RAlignedSquare1</i>	2D	•	•	AlignedSquare (2D) [self]
<i>RAlignedSquare2</i>	2D	•	•	AlignedSquare (2D) [self]

Canonical form is *RAlignedSquare1* (2D).

11.2.14 Square (2D)

A square.

Table 11.15: Square representations (2D)

Representation	Dim	In	Out	Inherited from
<i>RAlignedSquare1</i>	2D			AlignedSquare (2D)
<i>RAlignedSquare2</i>	2D			AlignedSquare (2D)
<i>RLinePoints</i>	2D	•	•	Square (2D) [self]
<i>RLineVector</i>	2D	•	•	Square (2D) [self]

Canonical form is *RLinePoints* (2D).

11.2.15 AlignedCube (3D)

A cube aligned with all axes.

Table 11.16: AlignedCube representations (3D)

Representation	Dim	In	Out	Inherited from
<i>RAlignedCube1</i>	3D	•	•	AlignedCube (3D) [self]
<i>RAlignedCube2</i>	3D	•	•	AlignedCube (3D) [self]

Canonical form is *RAlignedCube1* (3D).

11.2.16 Cube (3D)

A cube.

Table 11.17: Cube representations (3D)

Representation	Dim	In	Out	Inherited from
<i>RAlignedCube1</i>	3D			AlignedCube (3D)
<i>RAlignedCube2</i>	3D			AlignedCube (3D)
<i>RLinePoints</i>	3D	•	•	Cube (3D) [self]
<i>RLineVector</i>	3D	•	•	Cube (3D) [self]

Canonical form is *RLinePoints* (3D).

11.2.17 AlignedRectangle (2D)

A rectangle aligned with all axes.

Table 11.18: AlignedRectangle representations (2D)

Representation	Dim	In	Out	Inherited from
<i>RAlignedHalfAxes</i>	2D			AlignedEllipse (2D)
<i>RLinePoints</i>	2D	•	•	AlignedRectangle (2D) [self]
<i>RLineVector</i>	2D	•	•	AlignedRectangle (2D) [self]

Canonical form is *RLinePoints* (2D).

11.2.18 Rectangle (2D)

A rectangle.

Table 11.19: Rectangle representations (2D)

Representation	Dim	In	Out	Inherited from
<i>RAlignedHalfAxes</i>	2D			AlignedEllipse (2D)
<i>RCircle1</i>	2D			Circle (2D)
<i>RCircle2</i>	2D			Circle (2D)
<i>RCircle3</i>	2D			Circle (2D)
<i>RCircle4</i>	2D			Circle (2D)
<i>REllipseCovariance</i>	2D			Ellipse (2D)
<i>RHalfAxes</i>	2D			Ellipse (2D)
<i>RLinePoints</i>	2D			AlignedRectangle (2D), Line (2D)
<i>RLineVector</i>	2D			AlignedRectangle (2D), Line (2D)
<i>RPoint</i>	2D			Point (2D)
<i>RRectangle1</i>	2D	•	•	Rectangle (2D) [self]
<i>RRectangle2</i>	2D	•	•	Rectangle (2D) [self]

Canonical form is RRectangle1 (2D).

11.2.19 AlignedCuboid (3D)

A cuboid aligned with all axes.

Table 11.20: AlignedCuboid representations (3D)

Representation	Dim	In	Out	Inherited from
<i>RAlignedHalfAxes</i>	3D			AlignedEllipsoid (3D)
<i>RLinePoints</i>	3D	•	•	AlignedCuboid (3D) [self]
<i>RLineVector</i>	3D	•	•	AlignedCuboid (3D) [self]

Canonical form is RLinePoints (3D).

11.2.20 Cuboid (3D)

A cuboid.

Table 11.21: Cuboid representations (3D)

Representation	Dim	In	Out	Inherited from
<i>RAlignedHalfAxes</i>	3D			AlignedEllipsoid (3D)
<i>RCuboid1</i>	3D	•	•	Cuboid (3D) [self]
<i>RCuboid2</i>	3D	•	•	Cuboid (3D) [self]
<i>REllipsoidCovariance</i>	3D			Ellipsoid (3D)
<i>RHalfAxes</i>	3D			Ellipsoid (3D)
<i>RLinePoints</i>	3D			AlignedCuboid (3D), Line (3D)
<i>RLineVector</i>	3D			AlignedCuboid (3D), Line (3D)
<i>RPoint</i>	3D			Point (3D)
<i>RSphere1</i>	3D			Sphere (3D)
<i>RSphere2</i>	3D			Sphere (3D)
<i>RSphere3</i>	3D			Sphere (3D)
<i>RSphere4</i>	3D			Sphere (3D)
<i>RSphere5</i>	3D			Sphere (3D)

Canonical form is *RCuboid1* (3D).

11.2.21 Circle (2D)

A circle.

Table 11.22: Circle representations (2D)

Representation	Dim	In	Out	Inherited from
<i>RCircle1</i>	2D	•	•	Circle (2D) [self]
<i>RCircle2</i>	2D	•	•	Circle (2D) [self]
<i>RCircle3</i>	2D	•	•	Circle (2D) [self]
<i>RCircle4</i>	2D	•	•	Circle (2D) [self]
<i>RPoint</i>	2D			Point (2D)

Canonical form is *RCircle1* (2D).

11.2.22 Sphere (3D)

A sphere.

Table 11.23: Sphere representations (3D)

Representation	Dim	In	Out	Inherited from
<i>RPoint</i>	3D			Point (3D)
<i>RSphere1</i>	3D	•	•	Sphere (3D) [self]
<i>RSphere2</i>	3D	•	•	Sphere (3D) [self]
<i>RSphere3</i>	3D	•	•	Sphere (3D) [self]
<i>RSphere4</i>	3D	•	•	Sphere (3D) [self]
<i>RSphere5</i>	3D	•	•	Sphere (3D) [self]

Canonical form is *RSphere1* (3D).

11.2.23 AlignedEllipse (2D)

An ellipse aligned with all axes.

Table 11.24: AlignedEllipse representations (2D)

Representation	Dim	In	Out	Inherited from
<i>RAlignedHalfAxes</i>	2D	•	•	AlignedEllipse (2D) [self]
<i>RLinePoints</i>	2D			AlignedRectangle (2D)
<i>RLineVector</i>	2D			AlignedRectangle (2D)

Canonical form is *RAlignedHalfAxes* (2D).

11.2.24 Ellipse (2D)

An ellipse.

Table 11.25: Ellipse representations (2D)

Representation	Dim	In	Out	Inherited from
<i>RAlignedHalfAxes</i>	2D			AlignedEllipse (2D)
<i>RCircle1</i>	2D			Circle (2D)
<i>RCircle2</i>	2D			Circle (2D)
<i>RCircle3</i>	2D			Circle (2D)
<i>RCircle4</i>	2D			Circle (2D)
<i>REllipseCovariance</i>	2D	•	•	Ellipse (2D) [self]
<i>RHalfAxes</i>	2D	•	•	Ellipse (2D) [self]
<i>RLinePoints</i>	2D			AlignedRectangle (2D), Line (2D)
<i>RLineVector</i>	2D			AlignedRectangle (2D), Line (2D)
<i>RPoint</i>	2D			Point (2D)
<i>RRectangle1</i>	2D			Rectangle (2D)
<i>RRectangle2</i>	2D			Rectangle (2D)

Canonical form is *RHalfAxes* (2D).

11.2.25 AlignedEllipsoid (3D)

An ellipsoid aligned with all axes.

Table 11.26: AlignedEllipsoid representations (3D)

Representation	Dim	In	Out	Inherited from
<i>RAlignedHalfAxes</i>	3D	•	•	AlignedEllipsoid (3D) [self]
<i>RLinePoints</i>	3D			AlignedCuboid (3D)
<i>RLineVector</i>	3D			AlignedCuboid (3D)

Canonical form is *RAlignedHalfAxes* (3D).

11.2.26 Ellipsoid (3D)

An ellipsoid.

Table 11.27: Ellipsoid representations (3D)

Representation	Dim	In	Out	Inherited from
<i>RAignedHalfAxes</i>	3D			AlignedEllipsoid (3D)
<i>RCuboid1</i>	3D			Cuboid (3D)
<i>RCuboid2</i>	3D			Cuboid (3D)
<i>REllipsoidCovariance</i>	3D	•	•	Ellipsoid (3D) [self]
<i>RHalfAxes</i>	3D	•	•	Ellipsoid (3D) [self]
<i>RLinePoints</i>	3D			AlignedCuboid (3D), Line (3D)
<i>RLineVector</i>	3D			AlignedCuboid (3D), Line (3D)
<i>RPoint</i>	3D			Point (3D)
<i>RSphere1</i>	3D			Sphere (3D)
<i>RSphere2</i>	3D			Sphere (3D)
<i>RSphere3</i>	3D			Sphere (3D)
<i>RSphere4</i>	3D			Sphere (3D)
<i>RSphere5</i>	3D			Sphere (3D)

Canonical form is *RHalfAxes* (3D).

11.2.27 PolylineSpline (2D)

A set of connected splines (open).

Table 11.28: PolylineSpline representations (2D)

Representation	Dim	In	Out	Inherited from
<i>RPolylinePoints</i>	2D	•	•	PolylineSpline (2D) [self]
<i>RPolylineVector</i>	2D	•	•	PolylineSpline (2D) [self]

Canonical form is *RPolylineVector* (2D).

11.2.28 PolylineSpline (3D)

A set of connected splines (open).

Table 11.29: PolylineSpline representations (3D)

Representation	Dim	In	Out	Inherited from
<i>RPolylinePoints</i>	3D	•	•	PolylineSpline (3D) [self]
<i>RPolylineVector</i>	3D	•	•	PolylineSpline (3D) [self]

Canonical form is *RPolylineVector* (3D).

11.2.29 PolygonSpline (2D)

A set of connected splines (closed).

Table 11.30: PolygonSpline representations (2D)

Representation	Dim	In	Out	Inherited from
<i>RPolylinePoints</i>	2D	•	•	PolygonSpline (2D) [self]
<i>RPolylineVector</i>	2D	•	•	PolygonSpline (2D) [self]

Canonical form is *RPolylineVector* (2D).

11.2.30 PolygonSpline (3D)

A set of connected splines (closed).

Table 11.31: PolygonSpline representations (3D)

Representation	Dim	In	Out	Inherited from
<i>RPolylinePoints</i>	3D	•	•	PolygonSpline (3D) [self]
<i>RPolylineVector</i>	3D	•	•	PolygonSpline (3D) [self]

Canonical form is *RPolylineVector* (3D).

11.2.31 CircularCylinder (3D)

A circular cylinder.

Table 11.32: CircularCylinder representations (3D)

Representation	Dim	In	Out	Inherited from
<i>RCircularCylinder1</i>	3D	•	•	CircularCylinder (3D) [self]
<i>RCircularCylinder2</i>	3D	•	•	CircularCylinder (3D) [self]
<i>RCircularCylinder3</i>	3D	•	•	CircularCylinder (3D) [self]
<i>RCircularCylinder4</i>	3D	•	•	CircularCylinder (3D) [self]

Canonical form is *RCircularCylinder1* (3D).

11.2.32 EllipticCylinder (3D)

An elliptic cylinder.

Table 11.33: EllipticCylinder representations (3D)

Representation	Dim	In	Out	Inherited from
<i>RCircularCylinder1</i>	3D			CircularCylinder (3D)
<i>RCircularCylinder2</i>	3D			CircularCylinder (3D)
<i>RCircularCylinder3</i>	3D			CircularCylinder (3D)
<i>RCircularCylinder4</i>	3D			CircularCylinder (3D)
<i>REllipticCylinder1</i>	3D	•	•	EllipticCylinder (3D) [self]
<i>REllipticCylinder2</i>	3D	•	•	EllipticCylinder (3D) [self]
<i>REllipticCylinder3</i>	3D	•	•	EllipticCylinder (3D) [self]
<i>REllipticCylinder4</i>	3D	•	•	EllipticCylinder (3D) [self]

Canonical form is *REllipticCylinder1* (3D).

11.2.33 Arc (2D)

An arc.

Table 11.34: Arc representations (2D)

Representation	Dim	In	Out	Inherited from
<i>RArc1</i>	2D	•	•	Arc (2D) [self]
<i>RArc2</i>	2D	•	•	Arc (2D) [self]
<i>RArc3</i>	2D	•	•	Arc (2D) [self]

Canonical form is *RArc1* (2D).

11.2.34 Arc (3D)

An arc.

Table 11.35: Arc representations (3D)

Representation	Dim	In	Out	Inherited from
<i>RArc1</i>	3D	•	•	Arc (3D) [self]
<i>RArc2</i>	3D	•	•	Arc (3D) [self]
<i>RArc3</i>	3D	•	•	Arc (3D) [self]

Canonical form is *RArc1* (3D).

11.2.35 BitMask (1D)

A mask with one bit values.

Table 11.36: BitMask representations (1D)

Representation	Dim	In	Out	Inherited from
<i>RBitMask</i>	1D	•	•	BitMask (1D) [self]

Canonical form is *RBitMask* (1D).

11.2.36 BitMask (2D)

A mask with one bit values.

Table 11.37: BitMask representations (2D)

Representation	Dim	In	Out	Inherited from
<i>RBitMask</i>	2D	•	•	BitMask (2D) [self]

Canonical form is *RBitMask* (2D).

11.2.37 BitMask (3D)

A mask with one bit values.

Table 11.38: BitMask representations (3D)

Representation	Dim	In	Out	Inherited from
<i>RBitMask</i>	3D	•	•	BitMask (3D) [self]

Canonical form is RBitMask (3D).

11.2.38 GreyMask (1D)

A mask with multiple grey levels.

Table 11.39: GreyMask representations (1D)

Representation	Dim	In	Out	Inherited from
<i>RGreyMask</i>	1D	•	•	GreyMask (1D) [self]

Canonical form is RGreyMask (1D).

11.2.39 GreyMask (2D)

A mask with multiple grey levels.

Table 11.40: GreyMask representations (2D)

Representation	Dim	In	Out	Inherited from
<i>RGreyMask</i>	2D	•	•	GreyMask (2D) [self]

Canonical form is RGreyMask (2D).

11.2.40 GreyMask (3D)

A mask with multiple grey levels.

Table 11.41: GreyMask representations (3D)

Representation	Dim	In	Out	Inherited from
<i>RGreyMask</i>	3D	•	•	GreyMask (3D) [self]

Canonical form is RGreyMask (3D).

11.2.41 Mesh (2D)

A mesh.

Table 11.42: Mesh representations (2D)

Representation	Dim	In	Out	Inherited from
<i>RMesh</i>	2D	•	•	Mesh (2D) [self]

Canonical form is RMesh (2D).

11.2.42 Mesh (3D)

A mesh.

Table 11.43: Mesh representations (3D)

Representation	Dim	In	Out	Inherited from
<i>RMesh</i>	3D	•	•	Mesh (3D) [self]

Canonical form is RMesh (3D).

11.2.43 AffineTransform (1D)

Affine transformation of a shape.

Table 11.44: AffineTransform representations (1D)

Representation	Dim	In	Out	Inherited from
<i>RAffineTransform</i>	1D	•	•	AffineTransform (1D) [self]

Canonical form is RAffineTransform (1D).

11.2.44 AffineTransform (2D)

Affine transformation of a shape.

Table 11.45: AffineTransform representations (2D)

Representation	Dim	In	Out	Inherited from
<i>RAffineTransform</i>	2D	•	•	AffineTransform (2D) [self]

Canonical form is RAffineTransform (2D).

11.2.45 AffineTransform (3D)

Affine transformation of a shape.

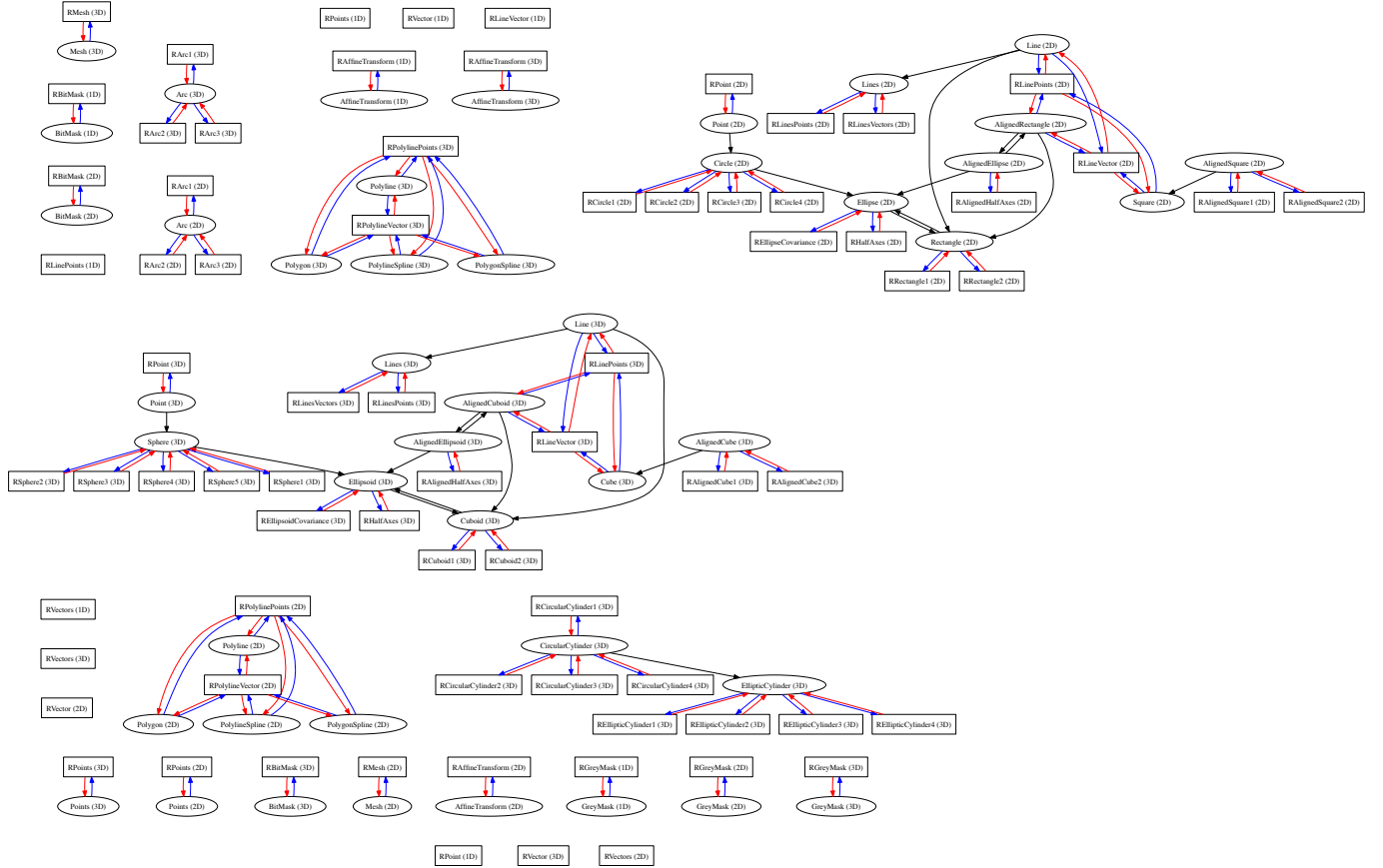
Table 11.46: AffineTransform representations (3D)

Representation	Dim	In	Out	Inherited from
<i>RAffineTransform</i>	3D	•	•	AffineTransform (3D) [self]

Canonical form is RAffineTransform (3D).

11.3 Relationships

The following figure illustrates the relationships detailed in the above tables. Ellipses are shapes, while representations are rectangles. Black arrows indicate inheritance of shape representations, while red and blue arrows indicate the representations possible to provide as input to and obtain as output from a shape, respectively.



SHAPE REPRESENTATIONS

12.1 Overview

Table 12.1: Representations

ID	Representation	Dims	Description
0	<i>RPoint</i>	1D	A single point in 1D
1	<i>RPoint</i>	2D	A single point in 2D
2	<i>RPoint</i>	3D	A single point in 3D
3	<i>RPoints</i>	1D	A list of points in 1D
4	<i>RPoints</i>	2D	A list of points in 2D
5	<i>RPoints</i>	3D	A list of points in 3D
6	<i>RVector</i>	1D	A vector in 1D
7	<i>RVector</i>	2D	A vector in 2D
8	<i>RVector</i>	3D	A vector in 3D
9	<i>RVectors</i>	1D	A list of vectors in 1D
10	<i>RVectors</i>	2D	A list of vectors in 2D
11	<i>RVectors</i>	3D	A list of vectors in 3D
12	<i>RLinePoints</i>	1D	A line described by two points in 1D
13	<i>RLinePoints</i>	2D	A line described by two points in 2D
14	<i>RLinePoints</i>	3D	A line described by two points in 3D
15	<i>RLineVector</i>	1D	A line described by a point and a vector
16	<i>RLineVector</i>	2D	A line described by a point and a vector
17	<i>RLineVector</i>	3D	A line described by a point and a vector
18	<i>RLinesPoints</i>	2D	A list of lines described by two points in 2D
19	<i>RLinesPoints</i>	3D	A list of lines described by two points in 3D
20	<i>RLinesVectors</i>	2D	A list of lines described by a point and a vector in 2D; can be used to represent a vector field
21	<i>RLinesVectors</i>	3D	A list of lines described by a point and a vector in 3D; can be used to represent a vector field
22	<i>RPolylinePoints</i>	2D	A list of points in a polyline in 2D [could use <i>RPoints2D</i> directly]
23	<i>RPolylinePoints</i>	3D	A list of points in a polyline in 3D [could use <i>RPoints3D</i> directly]
24	<i>RPolylineVector</i>	2D	A list of points in a polyline represented by a starting point and list of vectors in 2D

Continued on next page

Table 12.1 – continued from previous page

ID	Representation	Dims	Description
25	<i>RPolylineVector</i>	3D	A list of points in a polyline represented by a starting point and list of vectors in 3D
26	<i>RAlignedSquare1</i>	2D	A square in 2D aligned with the axes described by a corner point and adjacent corner
27	<i>RAlignedSquare2</i>	2D	A square in 2D aligned with the axes described by a corner point and vector to an adjacent corner
28	<i>RAlignedCube1</i>	3D	A cube in 3D aligned with the axes described by a corner point and adjacent corner
29	<i>RAlignedCube2</i>	3D	A cube in 3D aligned with the axes described by a corner point and vector to an adjacent corner
30	<i>RRectangle1</i>	2D	A rectangle in 2D described by two corner points and a vector
31	<i>RRectangle2</i>	2D	A rectangle in 2D described by a corner point and two vectors
32	<i>RCuboid1</i>	3D	A cuboid in 3D described by two adjacent corners and two vectors
33	<i>RCuboid2</i>	3D	A cuboid in 3D described by a corner and three vectors
34	<i>RCircle1</i>	2D	A circle in 2D described by a centre point and 1D radius
35	<i>RCircle2</i>	2D	A circle in 2D described by a centre point and 2D radius
36	<i>RCircle3</i>	2D	A circle in 2D described by a circumference point and vector to the centre point
37	<i>RCircle4</i>	2D	A circle in 2D described by three circumference points
38	<i>RSphere1</i>	3D	A sphere in 3D described by a centre point and 1D radius
39	<i>RSphere2</i>	3D	A sphere in 3D described by a centre point and 2D radius
40	<i>RSphere3</i>	3D	A sphere in 3D described by a centre point and 3D radius
41	<i>RSphere4</i>	3D	A sphere in 3D described by a surface point and vector to the centre point
42	<i>RSphere5</i>	3D	A sphere in 3D described by a four surface points
43	<i>RAlignedHalfAxes</i>	2D	An ellipse in 2D aligned with the axes described by two half axes
44	<i>RHalfAxes</i>	2D	An ellipse in 2D described by two half axes
45	<i>REllipseCovariance</i>	2D	An ellipse in 2D described by a centre point and covariance matrix
46	<i>RAlignedHalfAxes</i>	3D	An ellipsoid in 3D aligned with the axes
47	<i>RHalfAxes</i>	3D	An ellipsoid in 3D described by three half axes
48	<i>REllipsoidCovariance</i>	3D	An ellipsoid in 3D described by a centre point and covariance matrix
49	<i>RCircularCylinder1</i>	3D	A circular cylinder in 3D described by the centres of both faces and a radius

Continued on next page

Table 12.1 – continued from previous page

ID	Representation	Dims	Description
50	<i>RCircularCylinder2</i>	3D	A circular cylinder in 3D described by the centre of one face, vector to second face and a radius
51	<i>RCircularCylinder3</i>	3D	A circular cylinder in 3D with faces at different angles described by the centres of both faces and vectors specifying the radius and angles of the faces
52	<i>RCircularCylinder4</i>	3D	A circular cylinder in 3D with faces at different angles described by the centre of one face, vector to second face and vectors specifying the radius and angles of the faces
53	<i>REllipticCylinder1</i>	3D	An elliptic cylinder in 3D described by the centres both faces and half axes
54	<i>REllipticCylinder2</i>	3D	An elliptic cylinder in 3D described by the centre of one face, vector to second face and half axes
55	<i>REllipticCylinder3</i>	3D	An elliptic cylinder in 3D with faces at different angles described by the centres both faces and half axes and angles
56	<i>REllipticCylinder4</i>	3D	An elliptic cylinder in 3D with faces at different angles described by the centre of one face, vector to second face and half axes and angles
57	<i>RArc1</i>	2D	An arc in 2D described by a line (points) and vector
58	<i>RArc2</i>	2D	An arc in 2D described by a line (vector) and a vector
59	<i>RArc3</i>	2D	An arc in 2D described by three points; vector inferred from third point
60	<i>RArc1</i>	3D	An arc in 3D described by a line (points) and vector
61	<i>RArc2</i>	3D	An arc in 3D described by a line (vector) and a vector
62	<i>RArc3</i>	3D	An arc in 3D described by three points; vector inferred from third point
63	<i>RBitMask</i>	1D	A bitmask in 1D described by bounding line, dimensions and mask data
64	<i>RBitMask</i>	2D	A bitmask in 2D described by bounding rectangle, dimensions and mask data
65	<i>RBitMask</i>	3D	A bitmask in 3D described by bounding cuboid, dimensions and mask data
66	<i>RGreyMask</i>	1D	A greymask in 1D described by bounding line, dimensions and mask data
67	<i>RGreyMask</i>	2D	A greymask in 2D described by bounding rectangle, dimensions and mask data
68	<i>RGreyMask</i>	3D	A greymask in 3D described by bounding cuboid, dimensions and mask data
69	<i>RMesh</i>	2D	A face-vertex mesh in 2D described by face and vertex lists
70	<i>RMesh</i>	3D	A face-vertex mesh in 3D described by face and vertex lists
71	<i>RAffineTransform</i>	1D	An affine transform in 1D described by a transformation matrix and 1D shape to transform

Continued on next page

Table 12.1 – continued from previous page

ID	Representation	Dims	Description
72	<i>RAffineTransform</i>	2D	An affine transform in 2D described by a transformation matrix and 2D shape to transform
73	<i>RAffineTransform</i>	3D	An affine transform in 3D described by a transformation matrix and 3D shape to transform

12.2 Definitions

12.2.1 RPoint (1D)

A single point in 1D.

Table 12.2: RPoint members (1D)

SeqNo	Name	Type	Description
0	P1	Vertex1D	Point coordinate

12.2.2 RPoint (2D)

A single point in 2D.

Table 12.3: RPoint members (2D)

SeqNo	Name	Type	Description
0	P1	Vertex2D	Point coordinates

12.2.3 RPoint (3D)

A single point in 3D.

Table 12.4: RPoint members (3D)

SeqNo	Name	Type	Description
0	P1	Vertex3D	Point coordinates

12.2.4 RPoints (1D)

A list of points in 1D.

Table 12.5: RPoints members (1D)

SeqNo	Name	Type	Description
0	NPOINTS	Count	Number of points
1	POINTS	Vertex1D[NPOINTS]	Array of point coordinates

12.2.5 RPoints (2D)

A list of points in 2D.

Table 12.6: RPoints members (2D)

SeqNo	Name	Type	Description
0	NPOINTS	Count	Number of points
1	POINTS	Vertex2D[NPOINTS]	Array of point coordinates

12.2.6 RPoints (3D)

A list of points in 3D.

Table 12.7: RPoints members (3D)

SeqNo	Name	Type	Description
0	NPOINTS	Count	Number of points
1	POINTS	Vertex3D[NPOINTS]	Array of point coordinates

12.2.7 RVector (1D)

A vector in 1D.

Table 12.8: RVector members (1D)

SeqNo	Name	Type	Description
0	V1	Vector1D	Vector

12.2.8 RVector (2D)

A vector in 2D.

Table 12.9: RVector members (2D)

SeqNo	Name	Type	Description
0	V1	Vector2D	Vector

12.2.9 RVector (3D)

A vector in 3D.

Table 12.10: RVector members (3D)

SeqNo	Name	Type	Description
0	V1	Vector3D	Vector

12.2.10 RVectors (1D)

A list of vectors in 1D.

Table 12.11: RVectors members (1D)

SeqNo	Name	Type	Description
0	NVEC	Count	Number of vectors
1	VA1	Vector1D[NVEC]	Array of vectors

12.2.11 RVectors (2D)

A list of vectors in 2D.

Table 12.12: RVectors members (2D)

SeqNo	Name	Type	Description
0	NVEC	Count	Number of vectors
1	VA1	Vector2D[NVEC]	Array of vectors

12.2.12 RVectors (3D)

A list of vectors in 3D.

Table 12.13: RVectors members (3D)

SeqNo	Name	Type	Description
0	NVEC	Count	Number of vectors
1	VA1	Vector3D[NVEC]	Array of vectors

12.2.13 RLinePoints (1D)

A line described by two points in 1D.

Table 12.14: RLinePoints members (1D)

SeqNo	Name	Type	Description
0	P1	Vertex1D[2]	Line start and end points

12.2.14 RLinePoints (2D)

A line described by two points in 2D.

Table 12.15: RLinePoints members (2D)

SeqNo	Name	Type	Description
0	P1	Vertex2D[2]	Line start and end points

12.2.15 RLinePoints (3D)

A line described by two points in 3D.

Table 12.16: RLinePoints members (3D)

SeqNo	Name	Type	Description
0	P1	Vertex3D[2]	Line start and end points

12.2.16 RLineVector (1D)

A line described by a point and a vector.

Table 12.17: RLineVector members (1D)

SeqNo	Name	Type	Description
0	P1	Vertex1D	Line start

12.2.17 RLineVector (2D)

A line described by a point and a vector.

Table 12.18: RLineVector members (2D)

SeqNo	Name	Type	Description
0	P1	Vertex2D	Line start
1	V1	Vector2D	Line end (relative to P1)

12.2.18 RLineVector (3D)

A line described by a point and a vector.

Table 12.19: RLineVector members (3D)

SeqNo	Name	Type	Description
0	P1	Vertex3D	Line start
1	V1	Vector3D	Line end (relative to P1)

12.2.19 RLinesPoints (2D)

A list of lines described by two points in 2D.

Table 12.20: RLinesPoints members (2D)

SeqNo	Name	Type	Description
0	NLINES	Count	Number of lines
1	LINES	RLinePoints2D[NLINES]	Array of line points

12.2.20 RLinesPoints (3D)

A list of lines described by two points in 3D.

Table 12.21: RLinesPoints members (3D)

SeqNo	Name	Type	Description
0	NLINES	Count	Number of lines
1	LINES	RLinePoints3D[NLINES]	Array of line points

12.2.21 RLinesVectors (2D)

A list of lines described by a point and a vector in 2D; can be used to represent a vector field.

Table 12.22: RLinesVectors members (2D)

SeqNo	Name	Type	Description
0	NLINES	Count	Number of lines
1	LINES	RLineVector2D[NLINES]	Array of line vectors

12.2.22 RLinesVectors (3D)

A list of lines described by a point and a vector in 3D; can be used to represent a vector field.

Table 12.23: RLinesVectors members (3D)

SeqNo	Name	Type	Description
0	NLINES	Count	Number of lines
1	LINES	RLineVector3D[NLINES]	Array of line vectors

12.2.23 RPolylinePoints (2D)

A list of points in a polyline in 2D [could use RPoints2D directly].

Table 12.24: RPolylinePoints members (2D)

SeqNo	Name	Type	Description
0	P1	RPoints2D	Array of points

12.2.24 RPolylinePoints (3D)

A list of points in a polyline in 3D [could use RPoints3D directly].

Table 12.25: RPolylinePoints members (3D)

SeqNo	Name	Type	Description
0	P1	RPoints3D	Array of points

12.2.25 RPolylineVector (2D)

A list of points in a polyline represented by a starting point and list of vectors in 2D.

Table 12.26: RPolylineVector members (2D)

SeqNo	Name	Type	Description
0	P1	Vertex2D	First point
1	V1	RVectors2D	Array of vectors

12.2.26 RPolylineVector (3D)

A list of points in a polyline represented by a starting point and list of vectors in 3D.

Table 12.27: RPolylineVector members (3D)

SeqNo	Name	Type	Description
0	P1	Vertex3D	First point
1	V1	RVectors3D	Array of vectors

12.2.27 RAlignedSquare1 (2D)

A square in 2D aligned with the axes described by a corner point and adjacent corner.

Table 12.28: RAlignedSquare1 members (2D)

SeqNo	Name	Type	Description
0	P1	Vertex2D	First corner
1	P2	Vertex1D	x coordinate of adjacent/opposing corner

12.2.28 RAlignedSquare2 (2D)

A square in 2D aligned with the axes described by a corner point and vector to an adjacent corner.

Table 12.29: RAlignedSquare2 members (2D)

SeqNo	Name	Type	Description
0	P1	Vertex2D	First corner
1	P2	Vector1D	distance to adjacent corner on x axis (relative to P1)

12.2.29 RAlignedCube1 (3D)

A cube in 3D aligned with the axes described by a corner point and adjacent corner.

Table 12.30: RAlignedCube1 members (3D)

SeqNo	Name	Type	Description
0	P1	Vertex3D	First corner
1	P2	Vertex1D	x coordinate of adjacent/opposing corner

12.2.30 RAlignedCube2 (3D)

A cube in 3D aligned with the axes described by a corner point and vector to an adjacent corner.

Table 12.31: RAlignedCube2 members (3D)

SeqNo	Name	Type	Description
0	P1	Vertex3D	First corner
1	P2	Vector1D	distance to adjacent corner on x axis (relative to P1)

12.2.31 RRectangle1 (2D)

A rectangle in 2D described by two corner points and a vector.

Table 12.32: RRectangle1 members (2D)

SeqNo	Name	Type	Description
0	P1	Vertex2D	First corner
1	P2	Vertex2D	Adjacent corner
2	V1	Vector1D	Distance to corner opposing P1 (relative to P2)

12.2.32 RRectangle2 (2D)

A rectangle in 2D described by a corner point and two vectors.

Table 12.33: RRectangle2 members (2D)

SeqNo	Name	Type	Description
0	P1	Vertex2D	First corner
1	V1	Vector2D	Distance to adjacent corner (relative to P1)
2	V2	Vector1D	Distance to corner opposing P1 (relative to P2)

12.2.33 RCuboid1 (3D)

A cuboid in 3D described by two adjacent corners and two vectors.

Table 12.34: RCuboid1 members (3D)

SeqNo	Name	Type	Description
0	P1	Vertex3D	First corner
1	P2	Vertex3D	Second corner (adjacent to P1)
2	V1	Vector2D	Distance to third corner (adjacent to P2)
3	V2	Vector1D	Distance to fourth corner (opposing P1, adjacent to V1)

12.2.34 RCuboid2 (3D)

A cuboid in 3D described by a corner and three vectors.

Table 12.35: RCuboid2 members (3D)

SeqNo	Name	Type	Description
0	P1	Vertex3D	First corner
1	V1	Vector3D	Distance to second corner (relative to P1)
2	V2	Vector2D	Distance to third corner (relative to V1)
3	V3	Vector1D	Distance to fourth corner (relative to V2, opposing P1)

12.2.35 RCircle1 (2D)

A circle in 2D described by a centre point and 1D radius.

Table 12.36: RCircle1 members (2D)

SeqNo	Name	Type	Description
0	P1	Vertex2D	Centre point
1	V1	Vector1D	Radius

12.2.36 RCircle2 (2D)

A circle in 2D described by a centre point and 2D radius.

Table 12.37: RCircle2 members (2D)

SeqNo	Name	Type	Description
0	P1	Vertex2D	Centre point
1	V1	Vector2D	Radius

12.2.37 RCircle3 (2D)

A circle in 2D described by a circumference point and vector to the centre point.

Table 12.38: RCircle3 members (2D)

SeqNo	Name	Type	Description
0	P1	Vertex2D	Point on circumference
1	V1	Vector2D	Vector to centre

12.2.38 RCircle4 (2D)

A circle in 2D described by three circumference points.

Table 12.39: RCircle4 members (2D)

SeqNo	Name	Type	Description
0	P1	Vertex2D[3]	Three points on circumference

12.2.39 RSphere1 (3D)

A sphere in 3D described by a centre point and 1D radius.

Table 12.40: RSphere1 members (3D)

SeqNo	Name	Type	Description
0	P1	Vertex3D	Centre point
1	V1	Vector1D	Radius

12.2.40 RSphere2 (3D)

A sphere in 3D described by a centre point and 2D radius.

Table 12.41: RSphere2 members (3D)

SeqNo	Name	Type	Description
0	P1	Vertex3D	Centre point
1	V1	Vector2D	Radius

12.2.41 RSphere3 (3D)

A sphere in 3D described by a centre point and 3D radius.

Table 12.42: RSphere3 members (3D)

SeqNo	Name	Type	Description
0	P1	Vertex3D	Centre point
1	V1	Vector3D	Radius

12.2.42 RSphere4 (3D)

A sphere in 3D described by a surface point and vector to the centre point.

Table 12.43: RSphere4 members (3D)

SeqNo	Name	Type	Description
0	P1	Vertex3D	Point on surface
1	V1	Vector3D	Vector to centre

12.2.43 RSphere5 (3D)

A sphere in 3D described by a four surface points.

Table 12.44: RSphere5 members (3D)

SeqNo	Name	Type	Description
0	P1	Vertex3D[4]	Four points on surface

12.2.44 RAlignedHalfAxes (2D)

An ellipse in 2D aligned with the axes described by two half axes.

Table 12.45: RAlignedHalfAxes members (2D)

SeqNo	Name	Type	Description
0	P1	Vertex2D	Centre point
1	V1	Vector2D	Half axes (x,y)

12.2.45 RHalfAxes (2D)

An ellipse in 2D described by two half axes.

Table 12.46: RHalfAxes members (2D)

SeqNo	Name	Type	Description
0	P1	Vertex2D	Centre point
1	V1	Vector2D	Half axes (xy)
2	V1	Vector1D	Half axes (x)

12.2.46 REllipseCovariance (2D)

An ellipse in 2D described by a centre point and covariance matrix.

Table 12.47: REllipseCovariance members (2D)

SeqNo	Name	Type	Description
0	P1	Vertex2D	Centre point (mean)
1	COV1	double[2][2]	2 × 2 covariance matrix

12.2.47 RAlignedHalfAxes (3D)

An ellipsoid in 3D aligned with the axes.

Table 12.48: RAlignedHalfAxes members (3D)

SeqNo	Name	Type	Description
0	P1	Vertex3D	Centre point
1	V1	Vector3D	Half axes (x,y,z)

12.2.48 RHalfAxes (3D)

An ellipsoid in 3D described by three half axes.

Table 12.49: RHalfAxes members (3D)

SeqNo	Name	Type	Description
0	P1	Vertex3D	Centre point
1	V1	Vector3D	Half axes (xyz)
2	V2	Vector2D	Half axes (xy)
3	V3	Vector1D	Half axes (x)

12.2.49 REllipsoidCovariance (3D)

An ellipsoid in 3D described by a centre point and covariance matrix.

Table 12.50: REllipsoidCovariance members (3D)

SeqNo	Name	Type	Description
0	P1	Vertex3D	Centre point (mean)
1	COV1	double[3][3]	3 × 3 covariance matrix

12.2.50 RCircularCylinder1 (3D)

A circular cylinder in 3D described by the centres of both faces and a radius.

Table 12.51: RCircularCylinder1 members (3D)

SeqNo	Name	Type	Description
0	P1	Vertex3D	Centre of first face
1	P2	Vertex3D	Centre of second face
2	V1	Vector1D	Radius

12.2.51 RCircularCylinder2 (3D)

A circular cylinder in 3D described by the centre of one face, vector to second face and a radius.

Table 12.52: RCircularCylinder2 members (3D)

SeqNo	Name	Type	Description
0	P1	Vertex3D	Centre of first face
1	V1	Vector3D	Distance to centre of second face
2	V2	Vector1D	Radius

12.2.52 RCircularCylinder3 (3D)

A circular cylinder in 3D with faces at different angles described by the centres of both faces and vectors specifying the radius and angles of the faces.

Table 12.53: RCircularCylinder3 members (3D)

SeqNo	Name	Type	Description
0	P1	Vertex3D	Centre of first face
1	P2	Vertex3D	Centre of second face
2	V1	Vector3D	Radius and angle of first face
3	V2	Vector3D	Angle of second face

12.2.53 RCircularCylinder4 (3D)

A circular cylinder in 3D with faces at different angles described by the centre of one face, vector to second face and vectors specifying the radius and angles of the faces.

Table 12.54: RCircularCylinder4 members (3D)

SeqNo	Name	Type	Description
0	P1	Vertex3D	Centre of first face
1	V1	Vector3D	Distance to centre of second face
2	V2	Vector3D	Radius and angle of first face
3	V3	Vector3D	Angle of second face

12.2.54 REllipticCylinder1 (3D)

An elliptic cylinder in 3D described by the centres both faces and half axes.

Table 12.55: REllipticCylinder1 members (3D)

SeqNo	Name	Type	Description
0	P1	Vertex3D	Centre of first face
1	P2	Vertex3D	Centre of second face
2	V1	Vector2D	Half axes (xy)
3	V2	Vector1D	Half axes (x)

12.2.55 REllipticCylinder2 (3D)

An elliptic cylinder in 3D described by the centre of one face, vector to second face and half axes.

Table 12.56: REllipticCylinder2 members (3D)

SeqNo	Name	Type	Description
0	P1	Vertex3D	Centre of first face
1	V1	Vector3D	Distance to second face
2	V2	Vector3D	Half axes (xy)
3	V3	Vector2D	Half axes (x)

12.2.56 REllipticCylinder3 (3D)

An elliptic cylinder in 3D with faces at different angles described by the centres both faces and half axes and angles.

Table 12.57: REllipticCylinder3 members (3D)

SeqNo	Name	Type	Description
0	P1	Vertex3D	Centre of first face
1	P2	Vertex3D	Centre of second face
2	V1	Vector3D	Half axes of first face (xyz)
3	V2	Vector2D	Half axes of first face (xy)
4	V3	Vector3D	Angle of second face

12.2.57 REllipticCylinder4 (3D)

An elliptic cylinder in 3D with faces at different angles described by the centre of one face, vector to second face and half axes and angles.

Table 12.58: REllipticCylinder4 members (3D)

SeqNo	Name	Type	Description
0	P1	Vertex3D	Centre of first face
1	V1	Vector3D	Distance to second face
2	V2	Vector3D	Half axes (xyz)
3	V3	Vector2D	Half axes (xy)
4	V4	Vector3D	Angle of second face

12.2.58 RArc1 (2D)

An arc in 2D described by a line (points) and vector.

Table 12.59: RArc1 members (2D)

SeqNo	Name	Type	Description
0	P1	RLinePoints2D	Centre point and arc start
1	V1	Vector2D	Arc end

12.2.59 RArc2 (2D)

An arc in 2D described by a line (vector) and a vector.

Table 12.60: RArc2 members (2D)

SeqNo	Name	Type	Description
0	P1	RLineVector2D	Centre point and arc start
1	V1	Vector2D	Arc end

12.2.60 RArc3 (2D)

An arc in 2D described by three points; vector inferred from third point.

Table 12.61: RArc3 members (2D)

SeqNo	Name	Type	Description
0	P1	Vertex2D[3]	Centre point, arc start and arc end (vector inferred)

12.2.61 RArc1 (3D)

An arc in 3D described by a line (points) and vector.

Table 12.62: RArc1 members (3D)

SeqNo	Name	Type	Description
0	P1	RLinePoints3D	Centre point and arc start
1	V1	Vector3D	Arc end

12.2.62 RArc2 (3D)

An arc in 3D described by a line (vector) and a vector.

Table 12.63: RArc2 members (3D)

SeqNo	Name	Type	Description
0	P1	RLineVector3D	Centre point and arc start
1	V1	Vector3D	Arc end

12.2.63 RArc3 (3D)

An arc in 3D described by three points; vector inferred from third point.

Table 12.64: RArc3 members (3D)

SeqNo	Name	Type	Description
0	P1	Vertex3D[3]	Centre point, arc start and arc end (vector inferred)

12.2.64 RBitMask (1D)

A bitmask in 1D described by bounding line, dimensions and mask data.

Table 12.65: RBitMask members (1D)

SeqNo	Name	Type	Description
0	B1	RLinePoints2D	Bounding line
1	DIM1	Vector1D	Mask dimensions (x)
2	DATA	bool[x]	Mask data

12.2.65 RBitMask (2D)

A bitmask in 2D described by bounding rectangle, dimensions and mask data.

Table 12.66: RBitMask members (2D)

SeqNo	Name	Type	Description
0	B1	RLinePoints2D	Bounding box
1	DIM1	Vector2D	Mask dimensions (x,y)
2	DATA	bool[x,y]	Mask data

12.2.66 RBitMask (3D)

A bitmask in 3D described by bounding cuboid, dimensions and mask data.

Table 12.67: RBitMask members (3D)

SeqNo	Name	Type	Description
0	B1	RLinePoints3D	Bounding box
1	DIM1	Vector3D	Mask dimensions (x,y,z)
2	DATA	bool[x,y,z]	Mask data

12.2.67 RGreyMask (1D)

A greymask in 1D described by bounding line, dimensions and mask data.

Table 12.68: RGreyMask members (1D)

SeqNo	Name	Type	Description
0	B1	RLinePoints1D	Bounding line
1	DIM1	Vector1D	Mask dimensions (x)
2	DATA	double[x]	Mask data

12.2.68 RGreyMask (2D)

A greymask in 2D described by bounding rectangle, dimensions and mask data.

Table 12.69: RGreyMask members (2D)

SeqNo	Name	Type	Description
0	B1	RLinePoints2D	Bounding box
1	DIM1	Vector2D	Mask dimensions (x,y)
2	DATA	double[x,y]	Mask data

12.2.69 RGreyMask (3D)

A greymask in 3D described by bounding cuboid, dimensions and mask data.

Table 12.70: RGreyMask members (3D)

SeqNo	Name	Type	Description
0	B1	RLinePoints3D	Bounding box
1	DIM1	Vector3D	Mask dimensions (x,y,z)
2	DATA	double[x,y,z]	Mask data

12.2.70 RMesh (2D)

A face-vertex mesh in 2D described by face and vertex lists.

Table 12.71: RMesh members (2D)

SeqNo	Name	Type	Description
0	NFACE	Count	Number of faces
1	VREF	double[NFACE][3]	Vertex references per face, counterclockwise winding
2	NVERT	Count	Number of vertices
3	VERTS	Vertex2D[NVERT]	Vertex coordinates

12.2.71 RMesh (3D)

A face-vertex mesh in 3D described by face and vertex lists.

Table 12.72: RMesh members (3D)

SeqNo	Name	Type	Description
0	NFACE	Count	Number of faces
1	VREF	double[NFACE][3]	Vertex references per face, counterclockwise winding
2	NVERT	Count	Number of vertices
3	VERTS	Vertex3D[NVERT]	Vertex coordinates

12.2.72 RAffineTransform (1D)

An affine transform in 1D described by a transformation matrix and 1D shape to transform.

Table 12.73: RAffineTransform members (1D)

SeqNo	Name	Type	Description
0	TRANS	Affine1D	Transform
1	SHAPE	Shape1D	Shape

12.2.73 RAffineTransform (2D)

An affine transform in 2D described by a transformation matrix and 2D shape to transform.

Table 12.74: RAffineTransform members (2D)

SeqNo	Name	Type	Description
0	TRANS	Affine2D	Transform
1	SHAPE	Shape2D	Shape

12.2.74 RAffineTransform (3D)

An affine transform in 3D described by a transformation matrix and 3D shape to transform.

Table 12.75: RAffineTransform members (3D)

SeqNo	Name	Type	Description
0	TRANS	Affine3D	Transform
1	SHAPE	Shape3D	Shape

LAYERS

In the Zeiss AxioVision formats, ROIs (shapes) are contained within Layers. Sets of ROIs are collected in different layers. The UI only uses a single layer, but uses separate layers for acquisition and post-acquisition ROIs. But in the file format one may define arbitrary numbers of layers to act as a grouping mechanism for ROIs.

Adding layers as a top level grouping would permit related ROIs to be grouped together. However, this would also be possible using ROI-ROI links; it could be implemented using Layer-ROI links. Maybe layer could be a ROI type used solely for grouping?

ROI-ROI LINKS

ROI relationships: When segmenting cell contents, shown as cytoplasm, actin filaments, nucleus and nucleolus, these fall into a strict hierarchy (a nucleus can only be in one cell, though one cell could have more than one nucleus). If we added a ROI type that was a container of ROIs (note: not a union), and added a means of classifying ROIs with tags/labels, this would be very useful for HCS and other types of analysis. Additionally, some relationships are not hierarchical, e.g. tree-like branching and merging in a vessel bed, but could be represented if a ROI could point to one or more other ROIs, which would permit a directed graph of relationships between ROIs.

Tracking Containment User modification (branch/merge) Inherit properties Layer DAG

STORAGE OF VERTEX DATA

For quite a number of the shape primitives, it is possible to support 3D very simply—we just increase the number of dimensions in each vertex, and that’s it (obviously just for storage; it will still require some work for rendering). From the point of view of storing the list of vertices, it would be nice if we could specify the dimensions being used e.g. XZT, and then allow missing dimensions to be specified as constants as we now do for theZ. This will also mean that will be possible to use a 2D primitive with theZ set as equivalent to a 3D primitive with the z value specified separately to the (x,y) points. This would provide one means of keeping the representation compact. Additionally, it is undesirable to have a separate element for each vertex, since for complex shapes e.g. meshes, this would waste a lot of space: when scaling up to thousands of vertices, this would waste multi-megabytes of XML markup for no good reason.

15.1 XML schema

Shape type and representation are stored as unsigned 16 bit integers, counts as unsigned 32 bit integers, and vertices and vectors as double-precision floating point.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xsd:simpleType name="shapeDetailElement">
    <xsd:union memberTypes="xsd:unsignedShort xsd:unsignedInt xsd:double" />
  </xsd:simpleType>
  <xsd:simpleType name="shapeDetail">
    <xsd:list itemType="shapeDetailElement" />
  </xsd:simpleType>
  <xsd:element name="shape">
    <xsd:complexType>
      <xsd:simpleContent>
        <xsd:extension base="shapeDetail">
          </xsd:extension>
        </xsd:simpleContent>
      </xsd:complexType>
    </xsd:element>
  </xsd:schema>
```

15.2 Properties

Store at the level of the ROI, not the shape. Since all the shapes within a ROI describe a single entity, there is no need for separate properties (colour, line thickness/style/endings etc.) on each shape.

Note: **J-M Burel:** previous discussion about that. Need to review the notes taken at the time.

DEFINITION OF TERMS

ROI Region of interest. A subset of samples within an image. This is specified by the boundary or surface of the object.

Shape Geometric shape or mask. A shape is a geometric primitive or bitmask. A ROI is composed of one or more shapes.

INDICES AND TABLES

- *genindex*
- *search*

INDEX

Symbols

2D

Affine transform, 35

3D

Affine transform, 35

A

Affine

transform, 34

transform 2D, 35

transform 3D, 35

AffineTransform (1D), 50

AffineTransform (2D), 50

AffineTransform (3D), 50

AlignedCircularCylinder3D, 16

AlignedCube (3D), 41

AlignedCuboid (3D), 43

AlignedEllipse (2D), 44

AlignedEllipsoid (3D), 45

AlignedEllipsoid3D, 14

AlignedEllipticCylinder3D, 17

AlignedRectangle (2D), 42

AlignedSquare (2D), 41

Arc (2D), 47

Arc (3D), 48

Arc2D, 18

Arc3D, 19

Arcs, 18

B

BitMask (1D), 48

BitMask (2D), 48

BitMask (3D), 48

BitMask2D, 20

BitMask3D, 20

C

Circle (2D), 44

CircularCylinder (3D), 47

CircularCylinder3D, 16

Cube (3D), 42

Cuboid (3D), 43

Cylinders, 16

E

Ellipse (2D), 45

Ellipsoid (3D), 45

Ellipsoid3D, 14

EllipticCylinder (3D), 47

EllipticCylinder3D, 17

G

GreyMask (1D), 49

GreyMask (2D), 49

GreyMask (3D), 49

GreyMask2D, 19

GreyMask3D, 20

L

Labels, 21

Line (2D), 39

Line (3D), 39

Lines (2D), 39

Lines (3D), 40

M

Masks, 19

Mesh (2D), 49

Mesh (3D), 50

Mesh2D, 21

Mesh3D, 21

Meshes, 21

P

Point (2D), 38

Point (3D), 38

Points (2D), 38

Points (3D), 39

Polygon (2D), 40

Polygon (3D), 41

Polygon splines, [15](#)
PolygonSpline (2D), [46](#)
PolygonSpline (3D), [47](#)
PolygonSpline2D, [15](#)
PolygonSpline3D, [15](#)
Polyline (2D), [40](#)
Polyline (3D), [40](#)
Polyline Splines, [15](#)
PolylineSpline (2D), [46](#)
PolylineSpline (3D), [46](#)
PolylineSpline2D, [15](#)
PolylineSpline3D, [15](#)

R

RAffineTransform (1D), [71](#)
RAffineTransform (2D), [72](#)
RAffineTransform (3D), [72](#)
RAlignedCube1 (3D), [61](#)
RAlignedCube2 (3D), [62](#)
RAlignedHalfAxes (2D), [65](#)
RAlignedHalfAxes (3D), [65](#)
RAlignedSquare1 (2D), [61](#)
RAlignedSquare2 (2D), [61](#)
RArc1 (2D), [68](#)
RArc1 (3D), [69](#)
RArc2 (2D), [68](#)
RArc2 (3D), [69](#)
RArc3 (2D), [69](#)
RArc3 (3D), [69](#)
RBitMask (1D), [69](#)
RBitMask (2D), [70](#)
RBitMask (3D), [70](#)
RCircle1 (2D), [63](#)
RCircle2 (2D), [63](#)
RCircle3 (2D), [63](#)
RCircle4 (2D), [64](#)
RCircularCylinder1 (3D), [66](#)
RCircularCylinder2 (3D), [66](#)
RCircularCylinder3 (3D), [67](#)
RCircularCylinder4 (3D), [67](#)
RCuboid1 (3D), [62](#)
RCuboid2 (3D), [63](#)
Rectangle (2D), [42](#)
REllipseCovariance (2D), [65](#)
REllipsoidCovariance (3D), [66](#)
REllipticCylinder1 (3D), [67](#)
REllipticCylinder2 (3D), [67](#)
REllipticCylinder3 (3D), [68](#)
REllipticCylinder4 (3D), [68](#)
RGreyMask (1D), [70](#)
RGreyMask (2D), [70](#)
RGreyMask (3D), [71](#)
RHalfAxes (2D), [65](#)
RHalfAxes (3D), [66](#)
RLinePoints (1D), [58](#)
RLinePoints (2D), [58](#)
RLinePoints (3D), [58](#)
RLinesPoints (2D), [59](#)
RLinesPoints (3D), [59](#)
RLinesVectors (2D), [60](#)
RLinesVectors (3D), [60](#)
RLineVector (1D), [59](#)
RLineVector (2D), [59](#)
RLineVector (3D), [59](#)
RMesh (2D), [71](#)
RMesh (3D), [71](#)
ROI, [79](#)
RPoint (1D), [56](#)
RPoint (2D), [56](#)
RPoint (3D), [56](#)
RPoints (1D), [56](#)
RPoints (2D), [56](#)
RPoints (3D), [57](#)
RPolylinePoints (2D), [60](#)
RPolylinePoints (3D), [60](#)
RPolylineVector (2D), [60](#)
RPolylineVector (3D), [61](#)
RRectangle1 (2D), [62](#)
RRectangle2 (2D), [62](#)
RSphere1 (3D), [64](#)
RSphere2 (3D), [64](#)
RSphere3 (3D), [64](#)
RSphere4 (3D), [64](#)
RSphere5 (3D), [65](#)
RVector (1D), [57](#)
RVector (2D), [57](#)
RVector (3D), [57](#)
RVectors (1D), [57](#)
RVectors (2D), [58](#)
RVectors (3D), [58](#)

S

Scale bars, [22](#)
Scale2D, [22](#)
Scale3D, [22](#)
Shape, [12](#), **[79](#)**
Sphere (3D), [44](#)
Sphere3D, [13](#)
Square (2D), [41](#)

T

Text2D, [22](#)

transform

2D, Affine, [35](#)

3D, Affine, [35](#)

Affine, [34](#)

V

ValuenD, [25](#)