ROI Specification [DRAFT]

Release 0.0.1

Roger Leigh

October 04, 2012

CONTENTS

Contents:

CHAPTER

ONE

ROI DISCUSSION

1.1 ROIs in three dimensions

This is just a followup regarding discussion with J-M and Will earlier today, where we covered the possibility of adding support of 3D primitives to the model. (Note that opinions over what to add remain divided, and this certainly needs further discussion!) These are just my thoughts on how we might add initial support.

[Note that for the n-dimensional stuff, I just added it as something to think about–I'm not suggesting we add any support at this point.]

2D	3D	nD
Line [2DLine]	3DLine	nDLine
Rectangle	3DCube [Cube]	nDCube [Hypercube]
Ellipse	3DEllipse [Ellipsoid]	nDEllipse [Hyperellipsoid]
Point [2DPoint]	3DPoint	nDPoint
Mask	3DMask	
Path [2DPath]	3DPath	
Mesh [2DMesh]	3DMesh	
Text		

Currently, the ROI model only supports 2D shapes. In the above table, additional primitives for 3D (and nD) have been added. Due to the "3D" or "nD" prefix, these do not replace the existing 2D-only primitives for backward compatibility, and to make it clear that these are for 3D work. Note that the "nD" primitives would work in 2D, 3D and higher dimensions; the existing primitives could all be implemented in terms of nD primitives in the code, but it is useful to have fixed-dimension primitives in the model.

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. At a higher level, the same is implied for e.g. cell tracking in xyzt; being able to draw a single polyline line (or vector), rather than storing a single point or line at each timepoint results in us being able to compute velocity and direction changes directly–rather than having to compute this information from discrete shapes, the information is directly available in a single shape.

I've also noted that 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.

1.2 Basic 3D primitives

3DLine List of (x,y,z) vertices. Alternative representation: a single vertex and list of vectors.

nDLine List of e.g. (x,y,z,t) vertices (tracking movement including speed and direction changes). Alternative representation: a single vertex and list of vectors.

3DCube X,Y,Z,Width,Height,Depth The current representation is effectively a vertex and a vector. Alternative representation: Both Rectangle and 3DCube could be represented by two vertices.

nDCube Vertex + Vector Alternative representation: two vertices.

3DEllipse X,Y,Z,RadiusX,RadiusY,RadiusZ This representation is effectively a vertex and a vector. Alternative representations: - two vertices, - vertex + vector - single vertex and the Mahalanobis distance [most useful when computing distributions with covariance; enables rotation with n-1 degrees of freedom]

nDEllipse Same as for 3DEllipse alternative representations

3DPoint X,Y,Z

nDPoint X,Y,Z,...

1.3 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.

3DMask As for Mask, but in 3D. A 3DMesh could be computed from a 3DMask.

1.4 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.

1.5 Paths

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

1.6 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.

CHAPTER

TWO

2D PRIMITIVES IN 3D

2.1 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.

2.2 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 construct edfrom 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?

2.2.1 2D extrusion

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

2.2.2 2D decomposition

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

CHAPTER

THREE

GEOMETRIC SHAPE PRIMITIVES

3.1 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 dicussion. 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.

3.2 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 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.

3.3 Basic primitives

Basic primitives describing shape types, vertices and vectors:

Primitive	Representation	Description
ShapeID	uint16	Numeric shape identifier
RepID	uint16	Numeric shape representation identifier
Count	uint32	Number of objects
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

All shape primitives are described in terms of the above basic primitives. This means that all shape descriptions are serialisable as a list of double precision floating point values. It also means that for compatible shape types, the shape type may be changed while retaining the point list (e.g. polyline, polygon spline).

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).

Todo

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.

3.4 Shape serialisation

Key considerations:

- 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.

3.5 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

3.5.1 Point2D

Representation:

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex2D	Point coordinates

3.6 Points

A point is a single point in space.

3.6.1 Point2D

Representation:

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex2D	Point coordinates

3.6.2 Point3D

Representation:

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex3D	Point coordinates

3.6.3 Points2D

Representation:

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
NPOINTS	Count	Number of points
P1	Vertex2D	First point
...	Vertex2D	Further points
Pn	Vertex2D	Last point

3.6.4 Points3D

Representation:

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
NPOINTS	Count	Number of points
P1	Vertex3D	First point
...	Vertex3D	Further points
Pn	Vertex3D	Last point

3.7 Lines

A line is a single straight edge drawn between two points.

3.7.1 Line2D

Representation:

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex2D	Line start
P2	Vertex2D	Line end

3.7.2 Line3D

Representation:

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex3D	Line start
P2	Vertex3D	Line end

3.8 Distances

A distance is a vector describing the distance travelled from a starting point.

3.8.1 Distance2D

Representation:

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex2D	Line start
V1	Vector2D	Line end (relative to P1)

3.8.2 Distance3D

Representation:

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex3D	Line start
V1	Vector3D	Line end (relative to P1)

3.9 Polylines

3.9.1 Polyline2D

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

3.9.2 Polyline3D

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

3.10 Polygons

3.10.1 Polygon2D

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

3.10.2 Polygon3D

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

3.11 Polydistances

A polydistance is a series of vectors describing the series of distances travelled from a starting point.

3.11.1 Polydistance2D

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex2D	First point
NVEC	Count	Number of vectors
V1	Vector2D	Distance to second point (relative to P1)
...	Vector2D	Further distances
Vn	Vector2D	Last distance (relative to V(n-1))

3.11.2 Polydistance3D

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex3D	First point
NVEC	Count	Number of vectors
V1	Vector3D	Distance to second point (relative to P1)
...	Vector3D	Further distances
Vn	Vector3D	Last distance (relative to V(n-1))

3.12 Squares and rectangles

A square exists in its basic 2D form, and in the form of a cube in 3D. Non-square variants are the rectangle and cuboid. All have simplified aligned forms with the shape aligned to the axes.

3.12.1 AlignedSquare2D

Aligned at right angles to xy axes.

Representation 1: Vertex and point on x axis (y inferred).

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex2D	First corner
P2	Vertex1D	x coordinate of adjacent/opposing corner

Representation 2: Vertex and vector on x axis (y inferred).

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex2D	First corner
P2	Vector1D	distance to adjacent corner on x axis (relative to P1)

3.12.2 Square2D

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

Representation 1: Vertices of two opposing corners.

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex2D	First corner
P2	Vertex2D	Opposing corner

Representation 2: Vertex and vector to opposing corner.

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex2D	First corner
V1	Vector2D	Opposing corner (relative to P1)

3.12.3 AlignedCube3D

Aligned at right angles to xyz axes.

Representation 1: Vertex and point on x axis (y and z inferred).

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex3D	First corner
P2	Vertex1D	x coordinate of adjacent/opposing corner

Representation 2: Vertex and vector on x axis (y and z inferred).

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex3D	First corner
P2	Vector1D	distance to adjacent corner on x axis (relative to P1)

3.12.4 Cube3D

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

Representation 1: Vertices of two opposing corners.

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex3D	First corner
P2	Vertex3D	Opposing corner

Representation 2: Vertex and vector to opposing corner.

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex3D	First corner
V1	Vector3D	Opposing corner (relative to P1)

3.12.5 AlignedRectangle2D

Aligned at right angles to xy axes.

Representation 1: Two opposing corners.

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex2D	First corner
P2	Vertex2D	Opposing corner

Representation 2: Two opposing corners.

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex2D	First corner
V1	Vector2D	Distance to opposing corner (relative to P1)

3.12.6 Rectangle2D

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

Representation 1: P1 and P2 corners specify one edge; V1 specifies length of other edge.

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex2D	First corner
P2	Vertex2D	Adjacent corner
V1	Vector1D	Distance to corner opposing P1 (relative to P2)

Representation 2: Rotated, not aligned at right angles to xy axes. P1 is the first corner, V1 specifies the second corner and V2 the length of the other edge.

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex2D	First corner
V1	Vector2D	Distance to adjacent corner (relative to P1)
V2	Vector1D	Distance to corner opposing P1 (relative to P2)

3.12.7 AlignedCuboid3D

Aligned at right angles to xyz axes.

Representation 1: Two opposing corners.

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex3D	First corner
P2	Vertex3D	Opposing corner

Representation 2: Vertex and vector to opposing corner

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex3D	First corner
V1	Vector3D	Distance to opposing corner (relative to P1)

3.12.8 Cuboid3D

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

Representation 3: P1 and P2 corners specify one edge, V2 the corner to define the first 2D face, and V3 the corner to define the final two 2D faces, and opposes P1.

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex3D	First corner
P2	Vertex3D	Second corner (adjacent to P1)
V1	Vector2D	Distance to third corner (adjacent to P2)
V2	Vector1D	Distance to fourth corner (opposing P1, adjacent to V1)

Representation 4: P1 is the first corner, V1 specifies the second corner and V2 the corner to define the first 2D face, and V3 the corner to define the final two 2D faces, and opposes P1.

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex3D	First corner
V1	Vector3D	Distance to second corner (relative to P1)
V2	Vector2D	Distance to third corner (relative to V1)
V3	Vector1D	Distance to fourth corner (relative to V2, opposing P1)

3.13 Circles and ellipses

3.13.1 Circle2D

Representation 1: Centre point and radius (1D vector)

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex2D	Centre point
V1	Vector1D	Radius

Representation 2: Centre point and radius (2D vector)

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex2D	Centre point
V1	Vector2D	Radius

Representation: 3: Bounding square. Inherits all Square2D and AlignedSquare2D representations.

Todo

Specify using reversed radius (vector2D to centre) Specify using diameter (two points) Specify using three points around circumference (->radius and centre)

3.13.2 Sphere3D

Representation 1: Centre point and radius (1D vector)

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex3D	Centre point
V1	Vector1D	Radius

Representation 2: Centre point and radius (2D vector)

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex3D	Centre point
V1	Vector2D	Radius

Representation 3: Centre point and radius (3D vector)

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex3D	Centre point
V1	Vector3D	Radius

Representation: 4: Bounding cube. Inherits all Cube3D and AlignedCube3D representations.

Todo

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

3.13.3 AlignedEllipse2D

Aligned at right angles to xy axes.

Representation 1: Centre and half axes.

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex2D	Centre point
V1	Vector2D	Half axes (x,y)

Representation 2: Bounding rectangle. Inherits all AlignedRectangle2D representations.

3.13.4 Ellipse2D

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

Representation 1: Centre and half axes; V2 is at right-angles to V1, so has only one dimension.

Name	Type	Description
S1	ShapeID	Shape
R1	RepID	Representation
P1	Vertex2D	Centre point
V1	Vector2D	Half axes (xy)
V1	Vector1D	Half axes (x)

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

3.13.5 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.

3.13.6 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

3.14 Polyline Splines

3.14.1 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

3.14.2 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

3.15 Polygon splines

3.15.1 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

3.15.2 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

3.16 Cylinders

3.16.1 AlignedCircularCylinder3D

Aligned

3.16.2 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 cyclinders 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 cyclinders 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?

3.16.3 AlignedEllipticCylinder3D

Todo

Inherits from AlignedEllipse.

3.16.4 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 cyclinders 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.

3.17 Arcs

3.17.1 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

3.17.2 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

3.18 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.

3.18.1 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

3.18.2 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

3.18.3 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

3.18.4 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

3.19 Meshes

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

3.19.1 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.

3.19.2 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.

3.20 Labels

3.21 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.

3.21.1 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.

3.22 Scale bars

3.22.1 Scale2D

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

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

3.22.2 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?

3.23 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.

CHAPTER

FOUR

COMPOUND ROIS

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

4.1 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?

4.1.1 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
SHAPEn	Shape	Last shape

4.1.2 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

4.1.3 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

4.1.4 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

4.1.5 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's 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's 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.

4.2 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's 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 manupulation 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 doesn't sound too way out. But the current system is limited to storing only the first of these four contexts, which loses information. While it's 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.

CHAPTER

FIVE

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.

5.1 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.

CHAPTER

SIX

AFFINE TRANSFORMS

6.1 2D transforms

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

$$\begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix}$$

6.2 3D transforms

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

$$\begin{bmatrix} a & e & i & m \\ b & f & j & n \\ c & g & k & o \\ d & h & l & p \end{bmatrix}$$

CHAPTER

SEVEN

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 arbitary 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?

CHAPTER

EIGHT

ROI-ROI LINKS

ROI relationships: When segmenting cell contents, shown as cytoplasm, actin filaments, nucleus and nucleolus, these fall into a strict heirarchy (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 hierarchial, 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

CHAPTER

NINE

STORAGE OF VERTEX DATA

For quite a number of the shape primitives, it's 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 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.

9.1 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>
```

9.2 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.

CHAPTER

TEN

DEFINITION OF TERMS

ROI Region of interest. A subset of samples within a dataset. 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.

CHAPTER

ELEVEN

INDICES AND TABLES

- *genindex*
- *search*

INDEX

S

T