ROI Specification [DRAFT]

Release 0.0.1

Roger Leigh

September 28, 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

GEOMETRIC SHAPE PRIMITIVES

2.1 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. Additionally, 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.

2.2 Basic primitives

Basic primitives describing vertices and vectors:

Primitive	Representation	Description
Vertex1D	double	Vertex in 1D
Vertex2D	double, double	Vertex in 2D
Vertex3D	double, double, double	Vertex in 3D
Vector1D	double	Vector in 1D
Vector2D	double, double	Vector in 2D
Vector3D	double, double, double	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.

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).

2.3 Points

A point is a single point in space.

2.3.1 Point2D

Representation:

Name	Type	Description
P1	Vertex2D	Point coordinates

2.3.2 Point3D

Representation:

Name	Type	Description
P1	Vertex3D	Point coordinates

2.4 Lines

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

2.4.1 Line2D

Representation:

Name	Type	Description
P1	Vertex2D	Line start
P2	Vertex2D	Line end

2.4.2 Line3D

Representation:

Name	Type	Description
P1	Vertex3D	Line start
P2	Vertex3D	Line end

2.5 Distances

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

2.5.1 Distance2D

Representation:

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

2.5.2 Distance3D

Representation:

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

2.6 Polylines

2.6.1 Polyline2D

Name	Type	Description
P1	Vertex2D	Line start
P2	Vertex2D	Second point
...	Vertex2D	Further points
Pn	Vertex2D	Line end

2.6.2 Polyline3D

Name	Type	Description
P1	Vertex3D	Line start
P2	Vertex3D	Second point
...	Vertex3D	Further points
Pn	Vertex3D	Line end

2.7 Polygons

2.7.1 Polygon2D

Name	Type	Description
P1	Vertex2D	First vertex
P2	Vertex2D	Second vertex
...	Vertex2D	Further vertices
Pn	Vertex2D	Last vertex

2.7.2 Polygon3D

Name	Type	Description
P1	Vertex3D	First vertex
P2	Vertex3D	Second vertex
...	Vertex3D	Further vertices
Pn	Vertex3D	Last vertex

2.8 Polydistances

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

2.8.1 Polydistance2D

Name	Type	Description
P1	Vertex2D	First point
V1	Vector2D	Distance to second point (relative to P1)
V2	Vector2D	Distance to second point (relative to V1)
...	Vector2D	Further distances
Vn	Vector2D	Last distance (relative to V(n-1))

2.8.2 Polydistance3D

Name	Type	Description
P1	Vertex2D	First point
V1	Vector2D	Distance to second point (relative to P1)
V2	Vector2D	Distance to second point (relative to V1)
...	Vector2D	Further distances
Vn	Vector2D	Last distance (relative to V(n-1))

2.9 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.

2.9.1 Square2D

Representation 1: Aligned at right angles to xy axes. Vertex and point on x axis (y inferred).

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

Representation 2: Aligned at right angles to xy axes. Vertex and vector on x axis (y inferred).

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

Representation 3: Rotated, not aligned at right angles to xy axes.

Name	Type	Description
P1	Vertex2D	First corner
P2	Vertex2D	Opposing corner

Representation 4: Rotated, not aligned at right angles to xy axes.

Name	Type	Description
P1	Vertex2D	First corner
V1	Vector2D	Opposing corner (relative to P1)

2.9.2 Cube3D

Representation 1: Aligned at right angles to xyz axes. Vertex and point on x axis (y and z inferred).

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

Representation 2: Aligned at right angles to xyz axes. Vertex and vector on x axis (y and z inferred).

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

Representation 3: Rotated, not aligned at right angles to xy axes.

Name	Type	Description
P1	Vertex3D	First corner
P2	Vertex3D	Opposing corner

Representation 4: Rotated, not aligned at right angles to xy axes.

Name	Type	Description
P1	Vertex3D	First corner
V1	Vector3D	Opposing corner (relative to P1)

2.9.3 Rectangle2D

Representation 1: Aligned at right angles to xy axes. Two opposing corners.

[Aligned at right angles to xy axes] 1: P1 (Vertex2D), P2 (Vertex2D)

Two opposing corners

2: P1 (Vertex2D), V1 (Vector2D) Vertex and vector to opposing corner

[Rotated] 3: P1 (Vertex2D), P2 (Vertex2D), V1 (Vector1D)

P1 and P2 corners specify one edge; V1 specifies length of other edge

4: P1 (Vertex2D), V1 (Vector2D), V2 (Vector1D) P1 is the first corner, V1 specifies the second corner and V2 the length of the other edge.

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

Representation 2: Aligned at right angles to xy axes. Vertex and vector on x axis (y inferred).

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

Representation 3: Rotated, not aligned at right angles to xy axes.

Name	Type	Description
P1	Vertex2D	First corner
P2	Vertex2D	Opposing corner

Representation 4: Rotated, not aligned at right angles to xy axes.

Name	Type	Description
P1	Vertex2D	First corner
V1	Vector2D	Opposing corner (relative to P1)

2.9.4 Cuboid3D

[Aligned at right angles to xyz axes] 1: P1 (Vertex3D), P2 (Vertex3D)

Two opposing corners

2: P1 (Vertex3D), V1 (Vector3D) Vertex and vector to opposing corner

[Rotated] 3: P1 (Vertex3D), P2 (Vertex3D), V1 (Vector2D), V2 (Vector1D) 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.

4: P1 (Vertex3D), V1 (Vector3D), V2 (Vector2D), V3 (Vector1D) 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.

2.10 Circles and ellipses

2.10.1 Circle2D

1: P1 (Vertex2D), V1 (Vector1D) Centre and radius

2: P1 (Vertex2D), V1 (Vector2D) Centre and radius

3: All Square2D specifications Bounding square

2.10.2 Sphere3D

1: P1 (Vertex3D), V1 (Vector1D) Centre and radius

2: P1 (Vertex3D), V1 (Vector2D) Centre and radius

3: P1 (Vertex3D), V1 (Vector3D) Centre and radius

4: All Cube3D specifications Bounding cube

2.10.3 Ellipse2D

[Aligned at right angles to xy axes] 1: P1 (Vertex2D), V1 (Vector2D)

Centre and half axes

2: P1 (Vertex2D), V1 (Vector1D), V2 (Vector1D) Centre and half axes specified separately

3: All Rectangle2D (aligned at right-angle) specifications.

[Rotated] 4: P1 (Vertex2D), V1 (Vector2D), V2 (Vector1D)

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

5: All Rectangle2D (rotated) specifications. 6: P1 (Vertex2D) COV (double × 2^2)

Mahalanbobis distance used to draw an ellipse using the mean coordinates (P1) and 2×2 covariance matrix (COV)

2.10.4 Ellipsoid3D

[Aligned at right angles to xy axes] 1: P1 (Vertex3D), V1 (Vector3D)

Centre and half axes

2: P1 (Vertex2D), V1 (Vector1D), V2 (Vector1D), V3 (Vector1D) Centre and half axes specified separately

3: All Rectangle3D (aligned at right-angle) specifications.

[Rotated] 4: P1 (Vertex3D), V1 (Vector3D), V2 (Vector2D), V3 (Vector1D)

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

5: All Rectangle3D (rotated) specifications. 6: P1 (Vertex3D) COV (double × 3^2)

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

2.11 Polyline Splines

2.11.1 PolylineSpline2D

1: P1 (Vertex2D), P2 (Vertex2D), . . . , Pn (Vertex2D)

2.11.2 PolylineSpline3D

1: P1 (Vertex3D), P2 (Vertex3D), . . . , Pn (Vertex3D)

2.12 Polygon splines

2.12.1 PolygonSpline2D

1: P1 (Vertex2D), P2 (Vertex2D), . . . , Pn (Vertex2D)

2.12.2 PolygonSpline3D

1: P1 (Vertex3D), P2 (Vertex3D), . . . , Pn (Vertex3D)

2.13 Cylinders

2.13.1 Cylinder3D

[Circular] 1: P1 (Vertex3D), P2 (Vertex3D), V1 (Vector1D)

Start and endpoint, plus radius

2: P1 (Vertex3D), V1 (Vector3D), V2 (Vector1D) Start point, distance to endpoint, plus radius

3: P1 (Vertex3D), P2 (Vertex3D), V1 (Vector3D), V2 (Vector3D) 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.

3: P1 (Vertex3D), V1 (Vector3D), V2 (Vector3D), V3 (Vector3D) 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.

[Elliptic] 1: P1 (Vertex3D), P2 (Vertex3D), V1 (Vector2D), V2 (Vector1D)

Start and endpoint, plus half axes

2: P1 (Vertex3D), V1 (Vector3D), V2 (Vector2D), V3 (Vector1D) Start point, distance to endpoint, plus half axes

3: P1 (Vertex3D), P2 (Vertex3D), V1 (Vector3D), V2 (Vector2D) V3 (Vector3D) 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). Face angles other than right-angles let chains of cyclinders be used for tubular structures without gaps at the joins.

3: P1 (Vertex3D), V1 (Vector3D), V2 (Vector3D), V3 (Vector2D) V4 (Vector3D) Start point, distance to endpoint, plus vectors to define half axes (V2 and V3) and angle of start face, and unit vector defining angle of end face (V4). Face angles other than right-angles let chains of cyclinders be used for tubular structures without gaps at the joins.

2.14 Arcs

2.14.1 Arc2D

1: P1 (Vertex2D), P2 (Vertex2D), V1 (Vector2D) Two points and unit vector describe an arc

2: P1 (Vertex2D), V1 (Vector2D), V2 (Vector2D) Centre point, plus length and unit vector describe an arc

2.14.2 Arc3D

1: P1 (Vertex3D), P2 (Vertex3D), V1 (Vector3D) Two points and unit vector describe an arc

2: P1 (Vertex3D), V1 (Vector3D), V2 (Vector3D) Centre point, plus length and unit vector describe an arc

2.15 Masks

2.15.1 Mask2D

1: DIMS (Vector2D), OFFSET (Vector2D), DATA (double × (DIMS[0] × DIMS[1])) Dimensions specify the x and y size of the mask, and offset the offset of this mask into the plane; DATA should be stored outside the ROI specification either as BinData or (better) in an IFD for OME-TIFF.

2.15.2 Mask3D

1: DIMS (Vector3D), OFFSET (Vector3D), DATA (double × (DIMS[0] × DIMS[1] × DIMS[2]))
Dimensions specify the x, y and z size of the mask, and offset the offset of this mask into the volume; DATA should be stored outside the ROI specification either as BinData or (better) in a set of IFDs for OME-TIFF.

2.16 Meshes

2.16.1 Mesh2D

Representation depends on mesh format; shown here as face-vertex 1: NUMFACE (double), (V1REF (double), V2REF (double), V3REF (double)) × NUMFACE,

NUMVERT (double), V1 (Vertex2D) ... Vn (Vertex2D) Number of faces, followed by the three vertices (counterclockwise winding) for each face, number of vertices, followed by a list of vertices. Vertex-face mapping is implied.

2.16.2 Mesh3D

Representation depends on mesh format; shown here as face-vertex 1: NUMFACE (double), (V1REF (double), V2REF (double), V3REF (double)) × NUMFACE,

NUMVERT (double), V1 (Vertex3D) ... Vn (Vertex3D) Number of faces, followed by the three vertices (counterclockwise winding) for each face, number of vertices, followed by a list of vertices

Vertex-face mapping is implied.

2.17 Labels

2.17.1 Text2D

1: All Vertex2D and Vertex3D specifications Text aligned relative to a point

2: All Line2D and Line3D specifications Text aligned relative to a line

3: All Rectangle2D and Rectangle3D specifications Text aligned and flowed inside a rectangle

2.18 Scale bars

2.18.1 Scale2D

1: P1 (Vertex2D), P2 (Vertex2D) Scale bar with distance between the two points

2: P1 (Vertex2D), V1 (Vector2D) Scale bar with distance from the vector

2.18.2 Scale3D

1: P1 (Vertex3D), P2 (Vertex3D) Scale bar with distance between the two points

2: P1 (Vertex3D), V1 (Vector3D) Scale bar with distance from the vector

CHAPTER

THREE

AFFINE TRANSFORMS

3.1 2D transforms

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

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

3.2 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

FOUR

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

FIVE

INDICES AND TABLES

- *genindex*
- *search*

INDEX