JavaGeom: a java library for geometrical computations

December 22, 2008

JavaGeom is a library for creating geometrical objects (such as lines, circles, ellipses, polygons...), getting information on them (length of a curve, number of points of a polygon...) or between them (get intersection point of 2 lines...), and manipulating them (affine transforms, clipping, compositions...). The aim of the library is to provide a simple but efficient tool to be embed in applications requiring geometry calculations, e.g. vectorial drawing software, computer-aided design, dynamic geometry, or research programs.

The library is developed using the Java language.

CAUTION: this document is a set of working notes, and may be strongly outdated.

Contents

Chapter 1

Introduction

1.1 Library organization

1.1.1 Packages hierarchy

Packages are divided depending on the type of geoemtry used : geom2d for euclidean plane geometry, geom3d for euclidean space geometry, or geom3s for spherical geometry. Subpackages could include specialized shape types.

1.1.1.1 Subpackages for math.geom2d

curve hierarchy of curves and of domains

line straight lines and line segments

polygon domains bounded by straight objects

transform affine and vectorial transforms in the plane

conic circle, ellipse, parabola and hyperbola, and derived classes

1.1.2 Java version

Java 1.5 for first version.

Second version is planned to support generic collections, and to have a more detailed package subdivision: subpackages 'shapes', 'graph'. main package will contain interface hierarchy, utility classes, and utility implementations.

1.2 Conventions

We use the convention to call 'XXXSet' collections of objects which are not linked together, such as a set of points, and 'PolyXXX' collections of objects which form an other object, like Polyline2D.

Chapter 2

Base classes

2.1 Utility classes

Some classes which are not geometrical 'shapes' are needed to represent displacement, angles, or measures made on shapes.

2.1.1 Vector2D

Represents a translation in the plane, without changing orientation angle.

x translation in Ox direction

y translation in Oy direction

Vector2D(Point2D) constructor from a point

Vector2D(double,double) constructor from 2 shifts

2.1.2 PolarVector2D

A class which derives from Vector2D, which can specify the shift in polar coordinates (ρ and θ). Fields x and y are computed accordingly. Note: ρ can be negative.

2.1.3 Angle2D

This class is an utility class containing various methods useful when working with angles: add ad subtract angles, by keeping the result between 0 and 2π, checking if a angle is between 2 others.

2.1.4 Box2D

An isothetic rectangle. Inner fields: xmin, xmax, ymin and ymax.

2.2 Basic class hierarchy

Most classes in javaGeom2d implement the Shape2D interface, which derive in interfacs Point2D, Curve2D, and Domain2D.

2.2.1 Shape2D

A shape which can be drawn in the plane, i.e. an arbitrary set of points. Shapes are divided into 3 types, depending on their inner dimension: points (dimension 0), curves (dimension 1) and domains (dimension 2). Each type is represented by the corresponding interface.

The library handles unbounded shapes like straight lines, parabolas or hyperbolas. Some methods are provided to check if a given shape is bounded, and to clip it if this is not the case.

Methods are available to check if points belongs to the shape, and to compute distance from point to shape.

2.2.1.1 Abstract methods

contains(Point2D) checks if the shape contains a point.

getDistance(point) computes distance from a point to the shape.

isBounded() return true if the shape can be included in a box big enough. Unbounded shapes (such as straight lines, parabola, hyperbola...) need to be clipped before being displayed.

clip(Box2D) returns the parts of the shape which belong to the box. It is maybe better to provide some clipping methods in Box2D class.

transform(AffineTransform2D) return the shape obtained after the given affine transform. This is useful for designing drawing programs, for which one can rotate, resize or shear geometric primitives.

2.2.2 EmptySet2D

Special class which allows to return a result as a Shape2D, even when result is empty. For example: when computing intersection of 2 parallel lines. Need to decide when this class is useful, or if it can simply be replaced by a null reference.

2.3 Points

A point is defined by 2 coordinate x and y. Two points are equal if they have the same Cartesian coordinates.

2.3.1 Point2D

This is the base class, which extends java.geom.Point2D.Double. While Point2D is the base class of the geometry, it is preferable that each shape can interact with java Points as well.

2.3.2 AbstractPoint2D

A proposal for an interface, allowing to consider both PolarPoint2D and Point2D as implementation of the same interface. Another option is to consider Point2D as an interface, and implement as CartesianPoint2D and PolarPoint2D.

getX() returns the x coordinate

getY() returns the y coordinate

2.3.3 Point2D.INFINITY_POINT

This is a point located at the infinity (both coordinate are $+\infty$). It can be used for giving the result of geometric operations, such as the intersection point of 2 parallel lines. Useful ? -> consider it later, if want to consider projective geometry.

2.3.4 PointSet2D

A set of points. Distance to this shape is the distance to the nearest point of the set.

2.3.5 PolarPoint2D

A utility class, used for creating a point from a base point, and a polar vector.

2.4 Domains of the plane

All the shapes that can be filled. Hausdorff dimension of such shapes is 2. Boundary is a curve, implementing the Boundary2D interface.

2.4.1 Domain2D

The should be able to locate points (inside, outside, on the boundary), and to return their boundary.

2.4.1.1 Abstract methods

getBoundary():Boundary2D return the boundary curve or the set of boundary curves of the domain.

2.4.2 GenericDomain2D

This is a concrete implementation of a domain defined by an arbitrary boundary curve. Most computations are delegated to the boundary curve.

2.4.2.1 Constructor

Domain2D(Boundary2D) construction using a boundary as argument.

Chapter 3

Curves

3.1 Curve hierarchy

The hierarchy of curve interfaces and classes is illustrated on figure 3.1.

3.1.1 Curve2D

The generic interface for curves in the plane. It actually defines parametric curves. Each point of the curve can be identified by its curvilinear coordinate. This curvilinear coordinate belongs to an interval $[t_0; t_1]$, which can be known by using methods getT0() and getT1().

For non continuous curves, different parametrization. One possibility is to use $[0; 2n-1]$ where n is the number of curves. For $t \in [2k; 2k+1]$, $k = 0, 1, ...n-1$, the point belong to the k-st curve. For $t \in [2k+1; 2k+2]$, $k = 0, 1, ...n-2$, the point is the ending point of a curve, either the k-st or the $(k+1)$-st.

3.1.1.1 Abstract methods

getT0():double return the beginning of parameterization domain

getT1():double return the end of parameterization domain

getPoint(t):Point2D return the point corresponding to given parameter

getFirstPoint():Point2D return the first point of the curve. Can be an infinite point in the case of an infinite shape.

getLastPoint():Point2D return the last point of the curve. Can be an infinite point in the case of an infinite shape.

getPosition(point):double compute curve position of a point. Ideally, parameter is a point belonging to the curve, but it is possible to consider position of the projection of the point on the curve. Result is comprised in the interval given by t_0 and t_1.

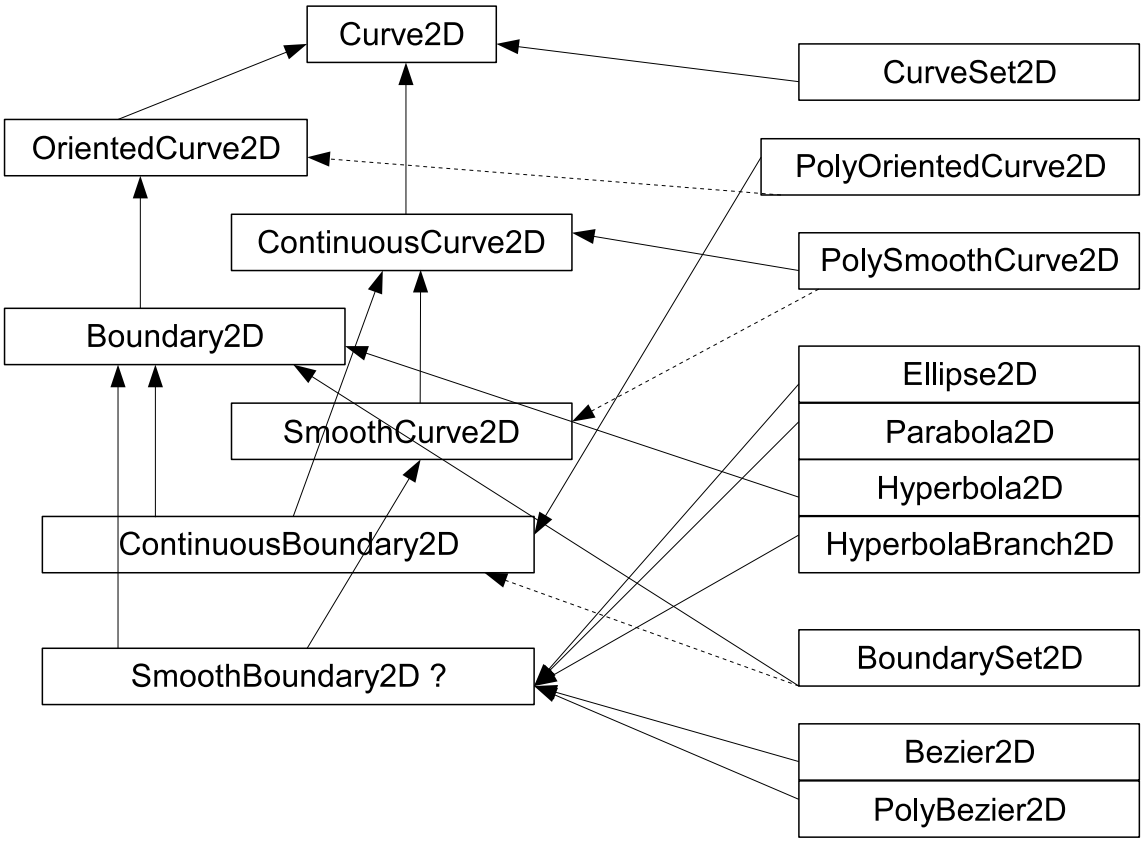

Figure 3.1: Hierarchy of curves

getDistance(Point2D),getDistance(double,double) return the distance to a point. Not always easy to implement (ex: conics), but this method is necessary to interactively select a curve.

getIntersections(LinearObject):Point2D[] return intersection points with a straight line, or a linear object. This function can be used to compute clipping with rectangle, or even polygons.

[deprecated]getSmoothPieces():SmoothCurve2D[] return smooth curves which compose this curve. Deprecated, as there is no special use of this function.

getContinuousCurves():ContinuousCurve[]

getReverseCurve() returns the same curve, but with inverted parametrization.

getSubCurve(t0,t1):Curve2D method of Curve2D or ContinuousCurve2D. Extract a portion of the curve.

3.1.2 ContinuousCurve2D

A curve that can be drawn without leaving the stroke out of the paper.

3.1.2.1 Abstract methods

getFirstTangent():Vector2D get the tangent at the first point (if defined)

getLastTangent():Vector2D get the tangent at the last point (if defined)

?getLeftTangent(t):Vector2D return the tangent vector when arriving on the point (not defined for the the first point of the curve)

?getRightTangent(t):Vector2D return the tangent vector when leaving the point (not defined for the the last point of the curve)

isClosed():boolean return true if the curve is closed (the first point is the same as the last point).

getSmoothPieces():SmoothCurve2D[] return the set of smooth curves composing the shape.

appendPath(GeneralPath):GeneralPath add the piece of curve to the path. If the curve is closed, it starts with a 'moveTo()' command. Anyway, it starts with a 'lineTo()', 'quadTo()' or 'cubicTo()'. This allows to concatenate easily continuous curves.

3.1.3 SmoothCurve2D

A curve without corners. This is the smallest unit to define curves.

A tangent line and a normal line are defined for each point of the curve. It is possible to draw an approximation of a thick curve by using tangent information on a set of points along the curve.

3.1.3.1 Abstract methods

getTangentVector(t):Vector2D get the tangent vector for the specified position

getCurvature(t):double returns the curvature for a point of the curve.

?getNormal(t):Vector2D get the normal vector. Not sure this is very useful ?

3.2 Oriented curves

In a boudary representation of planar shapes, curves are used to determine if point lie inside or outside the boundary. Several interfaces define the behaviour of boundary curves (such as straight lines, conics...), or of curves which can be used as a part of a boundary (such as line segments, coinic arcs...).

3.2.1 OrientedCurve2D

An OrientedCurve2D defines an 'inside' and an 'outside'. It is typically a part of the boundary of a domain. Several OrientedCurve2D chained together form a ContinuousBoundary2D. One or several ContinousBoundary2D form a Boundary2D.

If the curve is closed, the domain is definite. The same if the curve is infinite (parabola, line...). For open curves which are bounded, we can consider the extension of the curve by adding line arcs at the beginning and at the end of the curve, resulting in an infinite curve.

3.2.1.1 Abstract methods

getWindingAngle(Point):double return the angle portion the curve turn around the given point. Result is a signed angle.

getSignedDistance(Point):double signed distance from the point to the curve. Result is positive for a point outside the curve, negative for a point inside the curve, and zero for a point on the curve.

?getNormalCone(t):AngularWedge2D return the normal cone of the boundary, towards ouside. Should return an object AngularWedge2D, to be implemented, defined by a Point, a starting angle and a signed angular extent.

isInside(point):boolean return true if the point is located on the 'inside' part of the curve.

3.2.2 Boundary2D

A Boundary2D is used to describe the boundary of a Domain2D. It defines interior and exterior for every point of the plane. It extends OrientedCurve2D, and is composed of one or several ContinuousOrientedCurve2D. Each continuous boundary curve is either a closed curve, or an infinite curve.

3.2.2.1 Abstract methods

?getWindingNumber(Boundary2D,Point2D):double renvoie le nombre de fois où la courbe tourne autour du point. Peut être un nombre non entier (ex: une droite).

?getWindingNumber(Point2D) renvoie le nombre de fois où la courbe tourne autour du point. Peut être un nombre non entier (ex: une droite).

getBoundaryCurves():ContinuousOrientedCurve2D[] return a set of ContinuousBoundary2D.

?getClippedBoundary(Box):return a curve which is the boundary of the domain clipped by the given box.

3.2.3 ContinuousOrientedCurve2D

It is both a ContinuousCurve2D and an OrientedCurve2D. The reason of the existence of this class is to be able to separate the different components of a Boundary2D. A ContinuousBoundary2D is either bounded and closed, or an infinite curve, such as a straight line or a parabola.

3.2.4 ContinuousBoundary2D

same as ContinuousOrientedCurve2D, but implements Boundary2D.

3.2.5 SmoothOrientedCurve2D

A continuous boundary which is moreover smooth. Can be used as a shortcut for not specifying multiple interface, and to define a continuous oriented curve from multiple smooth oriented pieces. This is the smallest unit to define boundaries.

3.3 Some curve Implementations

3.3.1 CurveSet2D

A set of curves in the plane. Curves are not supposed to be contiguous.

As a CurveSet2D is an implementation of Curve2D, the composition can be recursive: a CurveSet2D contains one or several CurveSet2D, which contain also one or several CurveSet2D...

3.3.2 ContinuousCurveSet2D

Extends CurveSet2D, but each element of the set is a continuous curve. Some processing can be applied if the curve is an instance of a CurveSet2D.

3.3.3 ContinuousOrientedCurveSet2D

Extends CurveSet2D, but each element of the set is a continuous oriented curve.

3.3.4 BoundarySet2D

An implementation of a Boundary2D. This class provides an efficient way to define a planar domain by a set of ContinuousBoundary2D. This class has the same behaviour as CurveSet2D, but contains only ContinuousOrientedCurve2D. Name could be ContinuousOrientedCurveSet2D.

3.3.5 PolyCurve2D

Basically the same as a CurveSet2D, but it contains only continuous curves, and each curve is connected to the following one. This class implements ContinuousCurve2D. It is different from CurveSet2D, as curves are not linked in CurveSet2D. Maybe add an interface AbstractCurveSet2D, with getCurves() method ?

3.3.6 PolyOrientedCurve2D

A continuous curve, composed of smooth oriented curves. This class extends PolyCurve2D, and implements ContinuousOrientedCurve2D.

3.3.7 BoundaryPolyCurve2D

A continuous set of oriented curves, which defines a domain.

3.4 Algorithms on curves

3.4.1 Clip a Curve Set

Consider clipping with a box, i.e. an isothetic rectangle.

for a curve set, recursively clip each curve of the set, and add to a new CurveSet2D

3.4.2 Clip a Boundary2D

The same, but the result is a Boundary2D. need to take care of order of clipping.

Idea: compute intersections, order intersections, then link intersection on the edge of the box.

3.4.3 Point in PolyOrientedCurve2D

There are two possible algorithms: either use a variant of ray intersections count, or use orientation with respect to the closest oriented curve composing the set.

3.4.4 Length of a Curve

For many simple curves (e.g. circles, circle arcs, polylines), there is an explicit formula for computing there length. For a generic smooth curve, the use of a derivative allows to compute length of any curve by numerical integration. Need to specify tolerance limit.

Chapter 4

Lines and segments

4.1 Basic hierarchy

A straight object is an object which can be embedded in a Straight line. There are 3 straight objects in javaGeom: straight lines, line segments, and rays. A more generic class called 'LineArc2D' is used, which can be derived in each of these three classes.

4.1.1 LinearShape2D

The interface for all objects which can be embedded into a straight line. Maybe could be renamd as LinearShape2D ?

getSupportLine():StraightLine2D return the supporting line

4.1.2 AbstractLine2D

This abstract class gathers most of computation for classes like straight lines, line segments, and rays.

4.2 Implementations

4.2.1 StraightLine2D

A specialization of AbstractLine2D.

4.2.2 LineSegment2D

A specialization of AbstractLine2D.

4.2.3 Ray2D

A specialization of AbstractLine2D.

4.2.4 LineArc2D

It is defined from an origin, a vector, and the 2 limits of the parameterization.

4.2.5 LineObject2D

A line object is defined from 2 points. Properties of the object, such as vector or length, are computed from point references each time methods are called. This is a littl bit slower than line arcs, but allows to change line by changing the reference points.

Chapter 5

Polylines and polygons

5.1 Polylines

5.1.1 Polyline2D

Polyline is a set of points which describe a curve composed of several line segments. It is typically the boundary of a polygon.

getLineSegments()

getLength() sum of the lengths of each line segment.

5.1.2 ClosedPolyline2D

A specialization of PolyLine2D, dedicated to represente boundary of simple polygons.

5.2 Polygons

5.2.1 Polygon2D

Interface for polygons, which concerns all Domain2D whose boundary is composed uniquely of line segments.

getVertices():Collection<Point2D>

getEdges():Collection<LineSegment2D>

5.2.2 SimplePolygon2D

A polygon defined from an array of points. The boundary of a simple polygon is a closed polyline (a "ring" in JTS).

5.2.3 MultiplePolygon2D

A general Polygon 2D can be composed of several disjoint parts, and can contains holes. The boundary of a Polygon2D is a set of closed polylines.

5.3 Rectangles

5.3.1 Rectangle2D

Defined from a corner, width and length, and orientation.

5.3.2 CenteredRectangle2D

defined from a center, width and length, and orientation.

5.4 Other polygons

Square2D ? RegularPolygon2D ? IndexedPolygon2D ?

5.5 Algorithms on polygons

geometric operations: union, intersection, subtraction of polygons
- Minkowski sum of polygons
- computation of (signed) area and of perimeter length
- clipping of polygon
- triangulation of a polygon
- convex hull of a set of points.

Chapter 6

Circles, ellipses and other conics

6.1 Conics

The general equation for conics in the javaGeom2d libary has the following form:

$$ax^2 + bxy + cy^2 + dx + ey + f = 0$$

6.1.1 Conic types

Types of conics are ellipses, parabola and hyperbolas.

6.2 Ellipse

algebraic equation of an ellipse:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

6.3 Parabola

6.3.1 Representation

We use the following representation:

$$\begin{cases} x(t) &= t \\ y(t) &= at^2 \end{cases}, t \in \mathbb{R}$$

6.4 Hyperbola

composed of 2 branches

Algebraic equation of an hyperbola :

$$\frac{x^2}{a^2} - \frac{y^2}{b^2} = 1$$

6.5 Classes hierarchy

An interface Conic2D, and 3 subclasses : Ellipse2D, Parabola2D, Hyperbola2D. The class Circle2D inherits Ellipse2D. The class Hyperbola2D contains two instances of HyperbolaBranch2D.

Arcs of conics are implemented in 4 classes : EllipseArc2D, CircleArc2D, ParabolaArc2D and HyperbolaBranchArc2D.

6.6 Implementations

6.6.1 Conic2D

This is the interface for all conics. This interface provides methods for accessing conic type (ellipse, parabola), and conic parameter (center, focus, direction vectors...).

Conic2D is an extension of OrientedCurve2D, not of ContinuousCurve2D. This is due to the fact that hyperbolas are decomposed into 2 branches.

6.6.2 Ellipse2D

Implements ContinuousOrientedCurve2D, and SmoothCurve2D.

6.6.3 Circle2D

A circle inherits Ellipse2D.

6.6.4 Parabola2D

Implements ContinuousOrientedCurve2D, and SmoothCurve2D.

6.6.5 Hyperbola2D

An hyperbola is composed from 2 branches. Therefore, a general class Hyperbola2D can be defined as an implementation of OrientedCurve2D, which refers to 2 HyperbolaBranch2D, which are implementations of ContinuousOrientedCurve2D.

6.6.6 EllipseArc2D and CircleArc2D

A cheap implementation is to keep a reference to the supporting conic, and to add tests specific to angle management.

A reference to the support conic (ellipse or circle) is kept as member. Bounds of the arc are specified by the start angle and the angle extent. The arc is direct if the angle extent is positive. endAngle is simply startAngle+angleExtent. The bounds for parameterization are given by $t_0 = 0$ and $t_1 = |\text{angleExtent}|$.

An ellipse arc or circle arc can be created by specifying either start angle and angle extent, or bounding angles, but in the latter case one have to specify whether the arc is directed or not.

6.6.6.1 Constructors

EllipseArc(Ellipse, start, extent)

EllipseArc(Ellipse, start, end, direct)

EllipseArc(xc, yc, r1, r1, theta, start, extent)

EllipseArc(xc, yc, r1, r1, theta, start, end, direct)

Chapter 7

Polynomial curves

A quite common way used to describe abritrary curves. Different representations exist.

7.1 Representations

7.1.1 Polynomial curves

Parametrization of each coordinate is simply given by the suites of polynom coefficients.

7.1.2 Ferguson curves

They use Hermite representation: coordinate of extreme poitns, and derivatives. Usually defined for degree 3, coefficients for higher degrees are less intuitive.

7.1.3 Bezier Curves

Define a curve by several control points.

7.1.4 NURBS

Extension of Bezier curves, allow exact representation of conics.

7.2 Cubic Bezier curve

Currently only cubic Bezier curve defined by 4 control points is implemented.

Chapter 8

Transforms

Need to think a little bit further on transforms hierarchy, and on implementation of AffineTransform.

8.1 Planar Transforms

8.1.1 Translation2D

A translation is an isometry, a motion, and a direct transform.

If v_x and v_y are coordinates of the vector v, the transform is expressed as:

$$T_v = \begin{bmatrix} 1 & 0 & v_x \\ 0 & 1 & v_y \\ 0 & 0 & 1 \end{bmatrix}$$

The composition of wo translations with translation vectors (x_1, y_1) and (x_2, y_2) is another translation with translation vector $(x_1 + x_2, y_1 + y_2)$.

8.1.2 Rotation2D

A rotation is a motion, a direct transform and an isometry.

8.1.2.1 Matrix representation

If θ is the angle of rotation, the transform matrix is as follow:

$$R_\theta = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

For a rotation around a point $c = (c_x, c_y)$, the transforms becomes:

$$R_{c,\theta} = T_x R_\theta T_{-x} = \begin{bmatrix} \cos\theta & -\sin\theta & c_x(1-\cos\theta) + c_y \sin\theta \\ \sin\theta & \cos\theta & c_y(1-\cos\theta) - c_x \sin\theta \\ 0 & 0 & 1 \end{bmatrix}$$

8.1.2.2 Compositions

The composition with another rotation whith same center, is a new rotation, whose center is the same as the two rotations, and whose angle is the sum of the two rotation angles.

$$R_{c,\theta_1} \circ R_{c,\theta_2} = R_{c,\theta_1+\theta_2}$$

8.1.3 Homothecy2D

Is a similarity. Is direct if factor is positive.

Homothecy around origin by a factor s:

$$H_k = \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Homothecy by a factor s around a given point $c = (c_x, c_y)$:

$$H_{c,s} = T_x H_s T_{-x} = \begin{bmatrix} s & 0 & (1-s)\,c_x \\ 0 & s & (1-s)\,c_y \\ 0 & 0 & 1 \end{bmatrix}$$

8.1.4 Scaling2D

Similar to homothecy, but scaling coefficient is not the same in each direction. Then, this is not a similarity anymore.

For a scaling with coefficients $s_1$and s_2, matrix is expressed as

$$S_{c,s_x,s_y} = \begin{bmatrix} s_x & 0 & (1-s_x)\,c_x \\ 0 & s_y & (1-s_y)\,c_y \\ 0 & 0 & 1 \end{bmatrix}$$

8.1.5 LineReflection2D

Reflection about a line. Is an indirect isometry.

Note: to be implemented in StraightLine2D class.

8.1.6 PointReflection2D

Symmetry around a point. Is an isometry.

Note: to be implemented in Point2D class.

8.2 Transform interface hierarchy

8.2.1 Transform2D

can transform any point to another point, or an array of points. Preallocation must be considered for arrays, and differences between java and javaGeom kept in mind.

transformPoint(Point2D):Point2D

transformPoint(Point2D,Point2D):Point2D

8.2.2 Bijection2D

Transforms which can be inverted. interface which extends Transform2D.

getInverse():Bijection2D return the inverse transform

8.2.3 LinearTransform2D

Tranforms which can be represented with a matrix, such as affine transforms or projective transforms. Such transforms preserve alignment of points.

8.3 Implementation

8.3.1 AffineTransform2D

Tranforms which can be represented with a 3×3 matrix. Such transforms preserve parallelness of lines. A transform can be modified by concatenating with another transform.

AffineTransform2D is given as a class, and is immutable.

getMatrix():double[][]

abstract getCoefficients():double[]

transformVector(Vector2D):Vector2D

isMotion():boolean if transform is the composition of a translation and a rotation

isIsometry():boolean if transform keeps the unsigned area of the transformed shape unchanged.

isSimilarity():boolean if the transformed shape is the same as the original shapes, up to a scaling factor.

isDirect():boolean is true if transformed shapes keep the same orientation

getAsAwtTransform():java.awt.geom.AffineTransform

static createRotation(...):AffineTransform2D same as in java class

setToXXX(...) set transform to the transform XXX, with specific parameters.

8.4 Other transforms

8.4.1 ProjectiveTransform2D

Transforms a quadrilateral into another quadrilateral, without preserving parallelness. This is a superinterface of AffineTransform2D, and extends Bijection2D. In the case of affine transforms, it preserves parallelness. Can be represented with a matrix, but needs homogeneous coordinate.

getProjectiveMatrix return a 3×3 array of double

8.4.2 LinearTransform2D

A transform which can be represented using a matrix. A little bit more general than ProjectiveTransform, as perspective projections can also be represented using matrices.

8.4.3 CircleInversion2D

maybe subclass of linear transform. Think about it...

8.4.4 Projections

Project points on a line, a circle, or another shape. Some ones can be represented with a matrix (orthogonal projection). extends Transform2D, but not Bijection2D.

8.5 Algorithms for transforms

8.5.1 Computing inverse of a matrix

Suppose marix A is given in the form

$$A = \begin{bmatrix} m_{00} & m_{01} & m_{02} \\ m_{10} & m_{11} & m_{12} \\ 0 & 0 & 1 \end{bmatrix}$$

Then, inverse A^{-1} of matrix A is given by

$$A^{-1} = \frac{1}{m_{00}m_{11} - m_{01}m_{10}} \begin{bmatrix} m_{11} & -m_{01} & m_{01}m_{12} - m_{02}m_{11} \\ -m_{10} & m_{00} & m_{02}m_{10} - m_{00}m_{12} \\ 0 & 0 & 1 \end{bmatrix}$$

8.5.2 Test if affine transform is a similarity

We need to check if matrix is orthogonal. Apparently, It is enough to check that

$$m_{00}m_{01} + m_{10}m_{11} = 0$$

8.5.3 Test if affine transform is a motion or an isometry

First computes the determinant of the matrix. It equals 1 for a motion, and its absolute value equals 1 for an isometry.

Chapter 9

Planned extensions

9.1 Geometric Graphs

9.1.1 GeometricGraph2D

By 'geometric graph', we consider graphs whose vertices are points. It is an instance of Shape2D.

provides methods for accessing vertices and edges. maybe faces.

need exterior library for this. Jung ? JGraphT ?

getEdges():Collection<ContinuousCurve2D>

getVertices():Collection<Point2D>

9.2 Other models

9.2.1 Polynomial curves

Each coordinate is defined by a polynomial. Derivatives and normals are easy to calculate, but need maybe some library to handle polynomials.

9.2.2 Lissajous curves

Same as polynomial curves, except the model is different.

9.2.3 Fourier contour

Useful for studying the contour of real objects. Such model can have practical application.

9.3 Stochastic geometry

implement creation of random shapes, such as point processes, line processe, boolean models...

need efficient statistics library.

Idea is to create a stochastic process, whose realizations are shapes.

9.4 Other geometries

9.4.1 Spherical geometry

9.4.2 Hyperbolic geometry

9.4.3 Projection/Cartography

9.4.4 Projective geometry

9.5 Other shapes

9.5.1 Fractals

Need to have a more general interface system. Maybe as a different set, which can produce instances of Shape2D for different level of iteration (for iterated fractals).

9.5.2 Federer sets

9.5.3 Topological properties

This type of shapes should be able to make a difference between a set, its closure, and its interior.

9.5.4 Infinite spirals

Some bounded shapes can have infinite parametrization.