**ZSDraw**

Das Subsystem *ZSDraw* ist eine C++ Klassenbibliothek, um Zeichnungen bestehend aus zweidimensionalen, grafischen Objekten zu erstellen. Die einzelnen Objekte können im Simulationsmodus untereinander Daten austauschen und ihre Zustände zur Laufzeit ändern.

**Inhaltsverzeichnis**

[1. Einführung 3](#_Toc381771387)

[2. Grafische Objekte (Class CGraphObj) 4](#_Toc381771388)

[2.1. Koordinatensysteme 4](#_Toc381771389)

[2.1.1. Abbildungsvorschrift 4](#_Toc381771390)

[2.1.2. Mouse Events 7](#_Toc381771391)

# Einführung

# Grafische Objekte (Class CGraphObj)

## Koordinatensysteme

### Abbildungsvorschrift

(0/0)

x

y

#### Move (GraphicsItem.setPos)

(0/0)

x

y

(0/0)

scenePos.x

scenePos.y

#### Rotate (GraphicsItem.setTransform)

**transform.translate( -rctBounding.center )**

(0/0)

x

y

(0/0)

scenePos.x

scenePos.y

**transform.rotate( rotAngle\_deg )**

(0/0)

x

y

(0/0)

scenePos.x

scenePos.y

**transform.translate( rctBounding.center )**

(0/0)

x

y

(0/0)

scenePos.x

scenePos.y

### Mouse Events

#### Resizing Bounding Rectangle

The graphics item receives mouse press, mouse move and mouse release events. The mouse position is provided in scene coordinates, relative to the parent object (if any) and in item coordinates.

(0/0)

x

y

(0/0)

Item.

scenePos.x

Item

scenePos.y

mouseEv.scenePos

##### Calculating transformation values “straight forward” (wrong results)

**Calculate new Size**

(0/0)

x

y

Item.

scenePos.x

Item.

scenePos.y

mouseEv.pos

**Move (GraphicsItem.setPos)**

(0/0)

x

y

Item.

scenePos.x

Item.

scenePos.y

(0/0)

**Rotate.transform.translate( -rctBounding.center )**

(0/0)

x

y

Item.

scenePos.x

Item.

scenePos.y

(0/0)

**Roate.transform.rotate( rotAngle\_deg )**

(0/0)

x

y

Item.

scenePos.x

Item.

scenePos.y

(0/0)

**Rotate.transform.translate( rctBounding.center )**

(0/0)

x

y

Item.

scenePos.x

Item.

scenePos.y

(0/0)

mouseEv.scenePos

**Conclusion:**

Calculating the new transformation values “straight forward” leads to wrong results.

Resizing the original bounding rectangle, moving it to the “old” scene position, and rotating the object by transforming by center of resized bounding rectangle, applying the rotation angle and transforming back by center of resized bounding rectangle does not have the desired effect but the selection point of the object’s bounding rectangle would be moved away from the mouse position.

The calculation of the object’s transformation values have to be done in reverse order.

As you will see the bounding rectangle will be changed and the top left corner together with the bounding rectangle’s center point got to be calculated and corrected.

Depending on which selection points has been grabbed and to which position the mouse has been moved the left top corner of the rotated object may remain on the old position or may be moved together with the mouse. E.g. the left top corner itself may have been selected, the center point of the top line may be moved upwards, the center point of the left line may be moved to the left or any other shape point may be moved “beyond” the top left corner.

##### Moving bottom right corner right without passing top left corner

**Rotate.transform.translate( -rctBounding.center )**

The new “rctBounding.center()” has to be calculated by the mouse events scene position depending on the moved selection point. It has to be taken into account that the selection points could be moved to the “other side” of the opposite selection point (with a negative resulting rectangle size).

(0/0)

x

y

(0/0)

Item.

scenePos.x

Item

scenePos.y

mouseEv.scenePos

Calculating the new bounding rectangle’s center is easy if the scene position of the opposite selection point is known and is the center of the line from the selected point to the opposite selection point.

**Roate.transform.rotate( -rotAngle\_deg )**

(0/0)

x

y

(0/0)

Item.

scenePos.x

Item

scenePos.y

mouseEv.scenePos

**Rotate.transform.translate( rctBounding.center )**

(0/0)

x

y

(0/0)

Item.

scenePosOld.x

Item

scenePosOld.y

mouseEv.scenePos

Item.

scenePosNew.x

Item.

scenePosNew.y

The item’s scene position (the left top corner of the object’s bounding rectangle) has to be corrected depending on which selection point has been moved and to which position the mouse has been moved.

**Move (-GraphicsItem.setPos)**

(0/0)

x

y

(0/0)

Item.

scenePosOld.x

Item

scenePosOld.y

mouseEv.scenePos

Item.

scenePosNew.x

Item.

scenePosNew.y

##### Moving bottom right corner by passing top left corner

(0/0)

x

y

(0/0)

Item.

scenePos.x

Item

scenePos.y

mouseEv.

scenePos

(0/0)

**Detect whether top left corner has been passed in item’s coordinate system**

(0/0)

x

y

(0/0)

Item.

scenePos.x

Item

scenePos.y

mouseEv.pos

(0/0)

As the bottom right corner has been moved by mouse the opposite selection point of the bounding rectangle (the top left corner) has been passed if the mouse position’s X or Y coordinates (in item’s coordinate system) is less than zero. The new size of the bounding rectangle can be calculated.

**Rotate mouse position and opposite selection point around old center point**

(0/0)

x

y

(0/0)

Item.

scenePos.x

Item

scenePos.y

mouseEv.

scenePos

The top left corner of the rotated object becomes the new position (temporarily during mouse resize events) and the rotation point remains the same. The size of the object is calculated in item’s coordinate system. Depending on the size the shape points of polylines or child objects within groups) got to be newly set depending on their relative position to the top left corner of the original bounding rectangle.

On releasing the mouse and on pressing the mouse the normalized transformation values got to be calculated. The normalized transformation values are those which would be applied “straight” forward to the item in its coordinates system to transform it into the scene’s coordinate system.

To get the normalized transformation values a normal vector in the item’s coordinate system will be transformed into the scene’s coordinate system using the current transformation matrix of the graphics item. The move offset and the rotation angle can be determined this way. The x and y scale factors should be 1.0 as the item has been resized in the item’s coordinate system. Generally by resizing a graphics item using mouse events the “real” size of the item is modified and not the scale factors. By changing the scale factor of an item also the line width (the pen width) would be affected – and this is not desired.

(0/0)

x

y

Item.

scenePos.x

Item

scenePos.y

#### Moving Single Shape Point