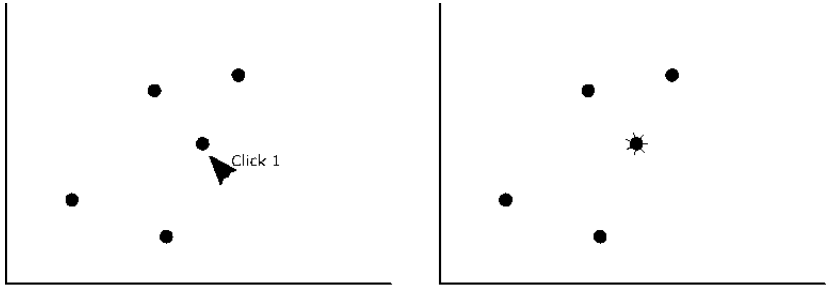
Selection Gestures

1. Point Selection



Steps to perform:

Point the cursor to a data point, then click

Procedural Steps:

Once(MouseClicked)

Pseudo Code:

```
point@1.location.select.data[item]
```

Application:

This gesture is used for selecting a single data point in a plot.

Example:

Since the gesture's function is general, it can be used in many occasions regarding data selection.

Usability:

This is the simplest gesture. The user just need to point to a data and then click it.

The potential problem is when there are more than one data in the same location. In this case, all of the data will be selected.

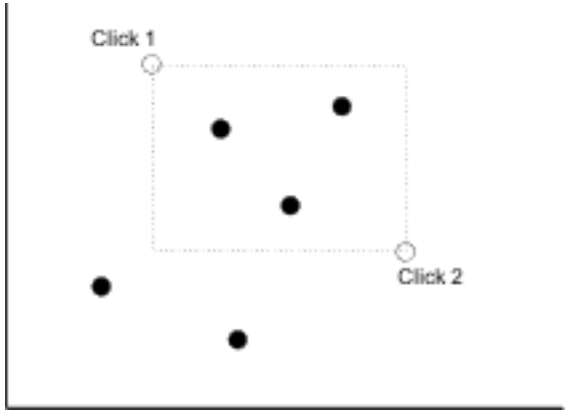
Performance complexity:

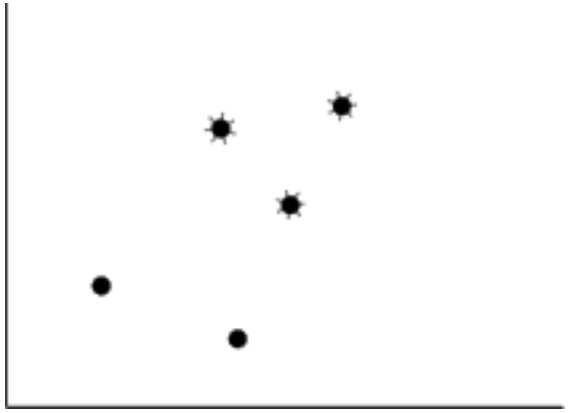
The user needs to click once for every single data. Therefore, it has O(N) complexity.

Variations:

None

2. Rubber-Band Selection





Steps to perform:

Clicking the mouse button once to select a corner, dragging to cover the data points, and then releasing the click to finish**.**

Procedural Steps:

Once(MousePressed), NoneOrMore(MouseDragged), Once(MouseReleased)

Pseudo Code:

rectangle(point@1,point@2).selectInside.data[item]

Application:

This gesture is used for selecting several data on a plot.

Example:

The gesture can be used in many occasions regarding multiple data selection.

Usability:

The potential problem is when there are data point(s) on the lasso line. In this case, all of the data will be selected.

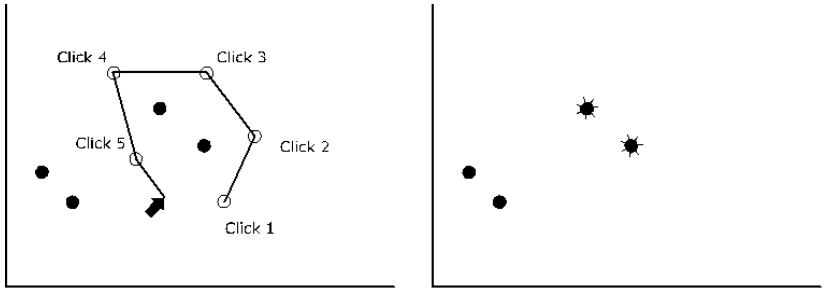
Performance complexity:

The user needs to make two clicks over data points. Since it has a fixed number of clicks over a number of data points, it has O(1) complexity.

Variations:

None

3. Lasso Selection



Steps to perform:

Multiple clicks around the data, right-click to finish

Procedural Steps:

Once(MouseLeftClick), NoneOrMore(MouseLeftClick), Once(MouseRightClick)

Pseudo Code:

```
points.polygon.selectInside.data[item]
```

Application:

This gesture is used for selecting several data on a plot.

Example:

The gesture can be used in many occasions regarding multiple data selection.

Usability:

The gesture will connect the first click to the last click to close the lasso.

The potential problem is when there are data point(s) on the lasso line. In this case, all of the data will be selected.

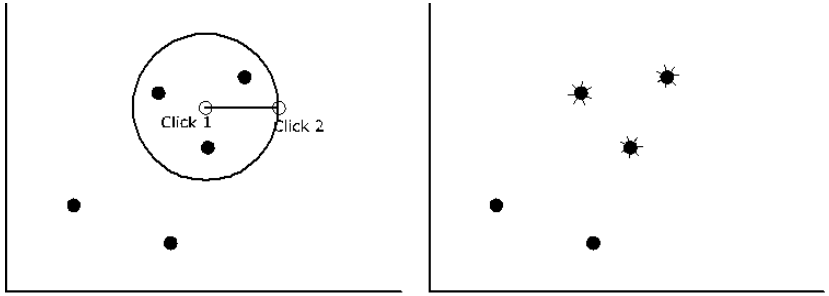
Performance complexity:

The user needs to make several clicks over data points. Since it has a comparable number of clicks over a number of data points, it has O(N) complexity.

Variations:

None

4. Circle Selection



Steps to perform:

The first click defines the center of the circle. The second click is the end of the gesture that defines the length of the radius.

Procedural Steps:

Once(MouseClick), Once(MouseClick)

Pseudo Code:

Circle(point@1, point@2).selectInside.data[item]

Application:

This gesture is used for selecting several data on a plot where there exists a circle or semi-circle geometry as the underlying shape of the data points.

Example:

Let us assume a scatter plot having any data. This gesture can be used to select data to perform various operations on the selected set of points.

Usability:

The user should make sure that the radius of the circle is not as long as the axises of the plot when being used for a scatter plot**.**

If the user wants to use ellipse geometry, s/he should use the next gesture.

Performance Complexity:

The user needs to make two clicks over data points. Since it has a fixed number of clicks over a number of data points, it has O(1) complexity.

Variations:

Using scroll to define the circle's radius

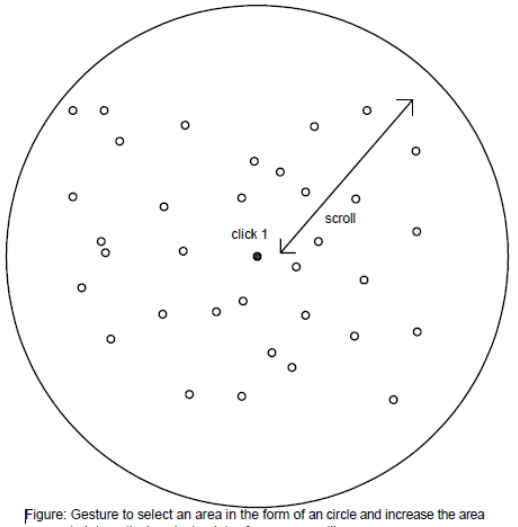
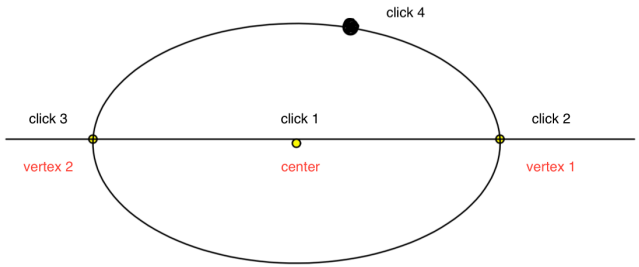


Figure: Gesture to select an area in the form of an circle and increase the area to interactively select points of some percentile.

5. Ellipse Selection



Steps to perform:

The first click defines the center of the ellipse. The second click will define the first focus of the ellipse. The third click will define the first vertex of the ellipse. A fourth click represents a point on the ellipse.

Procedural Steps:

Once(MouseClick), Once(MouseClick), Once(MouseClick), Once(MouseClick)

Pseudo Code:

ellipse(point@1, point@2).where(distancesum(distance(point@1,point@3) , distance(point@2,point@3))).map(distancesum(point@1,point@2)).selectInside.data[item]

Application:

This gesture can be used to select points on a plot using ellipse as the underlying geometry.

Example:

Let us assume a plot where we have a x-axis with values from -15 to +15 and and a y-axis where we have points from -5 to +5. To select space in a more optimal way we can use ellipse instead of a circle.

Usability:

The ellipse is drawn based on the first two points; the other two (focus and vertex) are mapped and following the formula where the distance between a point on the ellipse from both the focus is 2(a).

The fourth click itself is a form of interaction as it enables to user to interactively define the size of the ellipse.

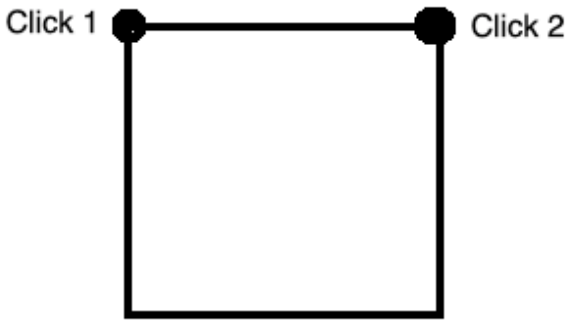
Performance Complexity:

Since it has a fixed number of clicks over a number of data points, it has O(1) complexity.

Variations:

The user can interactively change the dimensions of the co-vertex by clicking and dragging the co-vertex point on the y-axis. This will change the shape of the ellipse as the co-vertex is changed.

6. Square Selection



Steps to perform:

The first click defines the start of the square. The second click finished the gesture by defining the side of the square.

Procedural Steps:

Once(MouseClick), Once(MouseClick)

Pseudo Code:

square(point@1,point@2).selectInside.data[item]

Application:

This gesture can be used to select points over any type of plot.

Example:

Let us assume a scatter plot having some data points. This gesture can be used to select data using square as the underlying geometry.

Usability:

This gesture uses two points in a horizontal line to create the square. If the points are not located on a horizontal line, it will use the points as the diagonal points of the square.

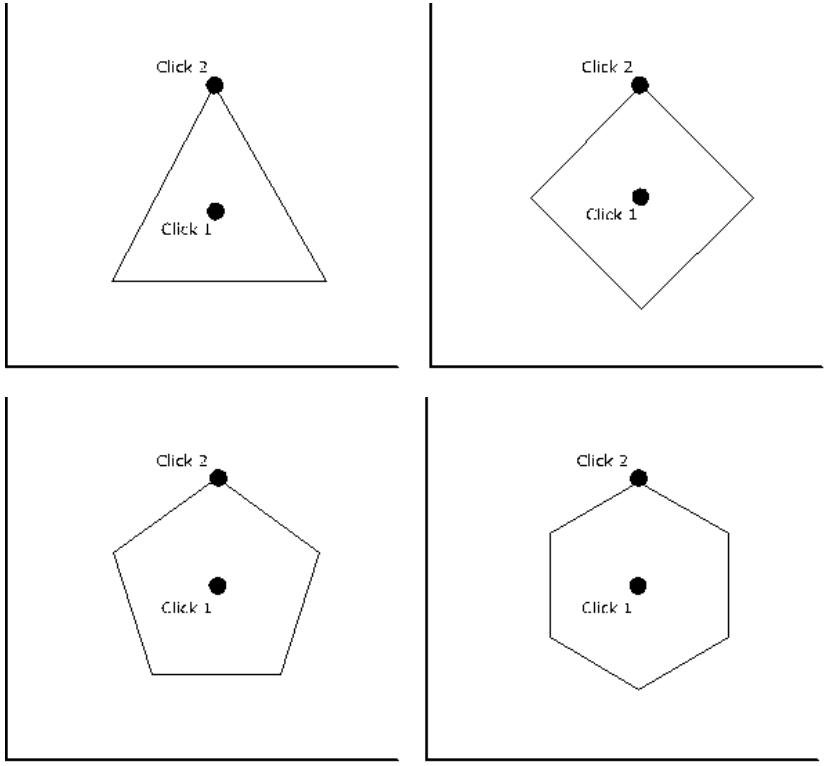
Performance Complexity:

Since it has a fixed number of clicks over a number of data points, it has O(1) complexity.

Variations:

None

7. Polygon Selection



Steps to perform:

The first click defines the center of the polygon. The second click is the end of the gesture that defines the length of the side.

Procedural Steps:

if(ControlDown), if(number[3-9]Down),Once(MouseClick), Once(MouseClick)

Pseudo Code:

```
Polygon(point@1, point@2,
number(numberPressed)).selectInside.data[item]
```

Application:

This gesture can be used to select all the points within the polygon over a plot.

Example:

Let us assume a scatter plot having some data points. This gesture can be used to select data based on the underlying polygon geometry.

Usability:

The second click specifies the top most vertex of the polygon. In the case where the polygon is not aligned to the axis of the plot, the user can adjust the position of the second click. The resulting polygon will be rotated accordingly.

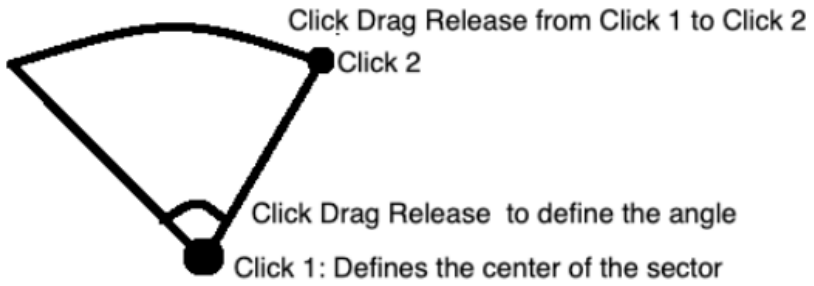
Performance Complexity:

Since it has a fixed number of clicks over a number of data points, it has O(1) complexity.

Variations:

None

8. Sector Selection



Steps to perform:

First click will define the center of the circle. The next set of click drag release will define the length of the radius. The final click drag and release will define the angle of the sector.

Procedural Steps:

Once(MouseClick), OnceOrMore(MouseDragged), Once(MouseReleased), Once(MouseClick), OnceOrMore(MouseDragged), Once(MouseReleased)

Pseudo Code:

```
Sector(point@1 ,
click.drag.release()).linesegment(point@1,point@2).
center(point@1).selectInside.data[item]
```

Application:

This gesture can be used to select points over any type of plot using sector as the underlying geometry.

Example:

Let us assume a scatter plot having clusters of data points. This gesture can be used to select a part of a cluster defined by certain location, angle, and radius.

Usability:

When there are data point(s) on the border line, all of the data will be selected.

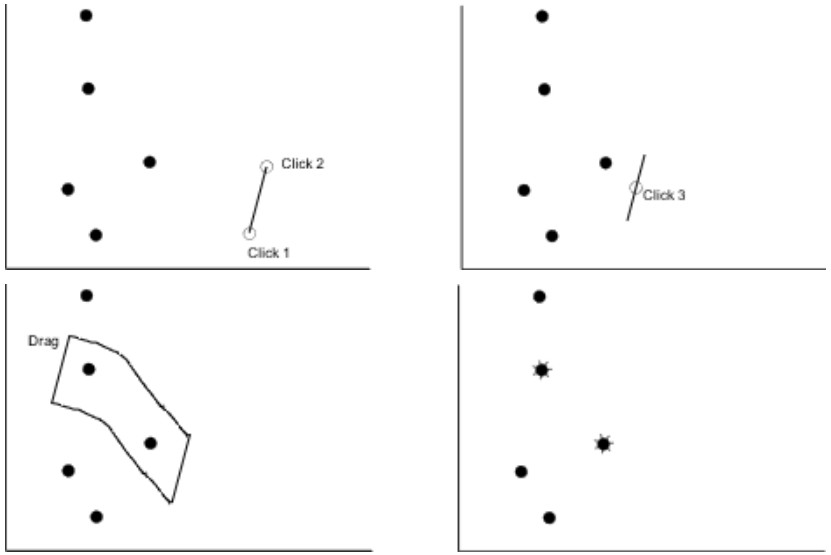
Performance Complexity:

Since it has a fixed number of clicks over a number of data points, it has O(1) complexity.

Variations:

None

9. Ribbon Selection



Steps to perform:

Two clicks to define the ribbon width and direction. Click to select a starting point, drag to cover the data points, release the click to finish

Procedural Steps:

Once(MouseClick), Once(MouseClick), Once(MouseClick), OnceOrMore(MouseDragged), Once(MouseReleased)

Pseudo Code:

```
line(point@1,point@2).swipe(event.drag()).selectInside.data[item]
```

Application:

This gesture is used for selecting several data on a plot by dragging the mouse to form a ribbon.

Example:

The gesture can be used in many occasions regarding multiple data selection where a ribbon is the underlying geometry.

Usability:

The potential problem is when the ribbon is too big for some locations. In this case, the user must redo the gesture from the beginning.

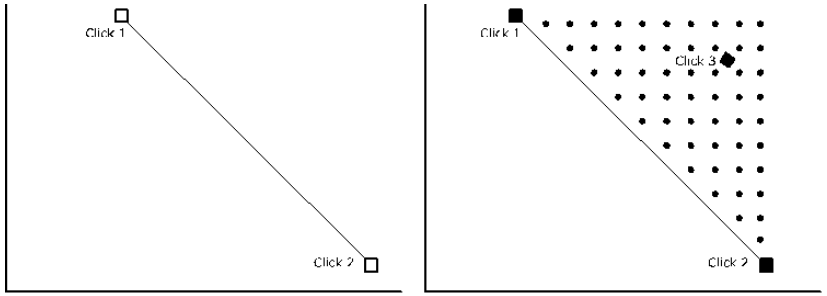
Performance complexity:

The user needs to make two clicks for the ribbon setup. Then, the user needs to drag the ribbon around to cover all data they want to select. It has roughly linear complexity O(N).

Variations:

None

10. Half Plane Selection



Steps to perform:

The first step is clicking any two points (click1 and click 2) in the space to define a line segment. Next, click any point (click3) in one of the planes to select that area.

Procedural Steps:

Once(MouseClick), Once(MouseClick), Once(MouseClick)

Pseudo Code:

```
plane(point@3,
line(point@1,point@2)).selectInside.data[item]
```

Application:

This is a general-purpose gesture that can be used to select areas divided by a line.

Example:

This gesture can be used by a civil surveyor to specify the area that falls to the side of a road. This gesture can also be used to partition the search space while looking up artifacts by an archeologist and break his work.

Usability:

This gesture has the ability to select all data points without putting the data inside a closed polygon, i.e. it has infinite geometry as its working shape. This feature is handy if there are a lot of data that need to be selected, spanning multiple pages of the view.

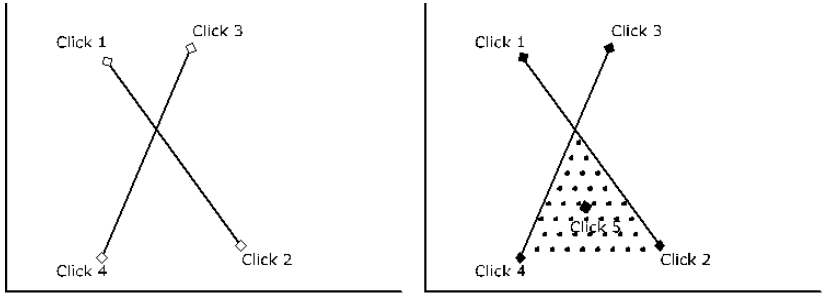
Performance Complexity:

Since it has a fixed number of clicks over a number of data points, it has O(1) complexity.

Variations:

None

11. Quarter Plane Selection



Steps to perform:

The first step is clicking any two points (click1 and click 2) in the space to define a line segment. Next, click two points (click3 and click4) on the plane to form a line segment intersecting the line segment drawn in step 1. Lastly, click any point (click5) in the space to select an area that falls on any of the four sides that are formed by the intersecting lines.

Procedural Steps:

Once(MouseClick), Once(MouseClick), Once(MouseClick), Once(MouseClick), Once(MouseClick)

Pseudo Code:

```
plane(point@5, lines(line(point@1, point@2), line.
(point@3, point@4).intersect(line(point@1,
point@2)))).selectInside.data[item]
```

Application:

This is a gesture that can be used to select an area defined by two line segments.

Example:

This gesture can be used to identify the categorical data like the theme of the books in a library, the genre of music in a collection, the variety of crops in a location, the categories of archeological specimens found in a site.

Usability:

This gesture has the same properties as Half Plane gesture, with addition of the ability to select a more specific region.

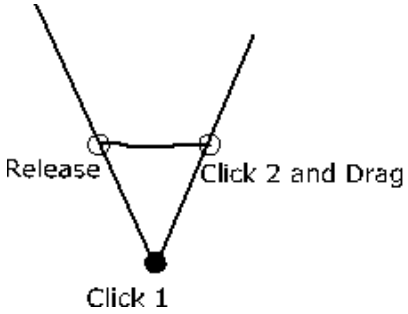
Performance Complexity:

Since it has a fixed number of clicks over a number of data points, it has O(1) complexity.

Variations:

None

12. Cone Selection



Steps to perform:

The first click defines the center of the cone. Then, the user clicks, drags, and releases the mouse button on top of the center to define the angle of the cone.

Procedural Steps:

Once(MouseClick), Once(MousePressed), NoneOrMore(MouseDragged), Once(MouseReleased)

Pseudo Code:

```
Angle(point@1, point@2,
click.drag.release()).selectInside.data[item]
```

Application:

This gesture can be used to select all the points within the cone in a plot.

Example:

Let us assume a scatter plot having some data points. This gesture can be used to select data using cone as the underlying geometry.

Usability:

The same as with Half Plane gesture, Cone Selection has the ability to select all data points without putting the data inside a closed polygon, i.e. it has infinite geometry as its working shape. This feature is handy if there are a lot of data that need to be selected, spanning multiple pages of the view.

The user should make sure to draw the angle with its inside facing the center.

Performance Complexity:

The user needs to make two clicks and a drag to cover all data they want to select. It has roughly linear complexity O(N).

Variations:

Instead of using click-and-drag, we can use three clicks

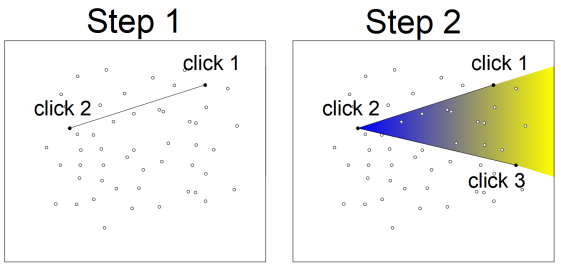
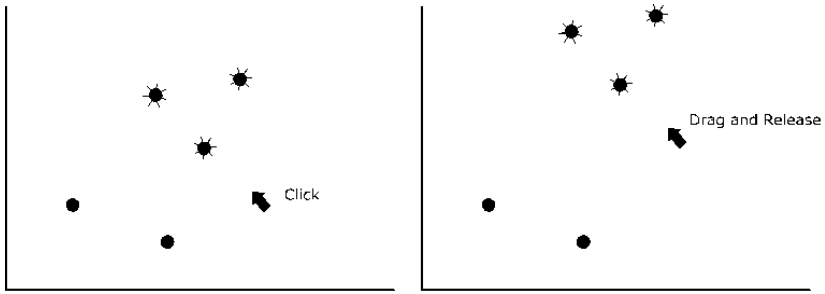


Figure: Gesture to select area based on cone. step1:Define first line(click1,click2) step2:define second line (click3)

Navigation Gestures

1. Panning



Steps to perform:

First click on the plane, then drag and release the cursor to a new location.

Procedural Steps:

Once(MousePressed), NoneOrMore(MouseDragged), Once(MouseReleased)

Pseudo Code:

```
location.selectedData[item].move(point@1,point@2)
```

Application:

This gesture is used for moving selected data points to a new location.

Example:

Since the gesture's function is general, it can be used in many occasions regarding data movement.

Usability:

This is a simple gesture. The user just need to click on the plot area and then drag the cursor to the target location.

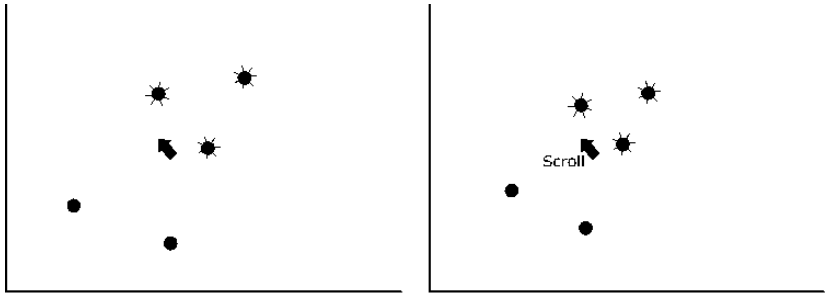
Performance complexity:

The user needs to click once and then drag the cursor to move all of the selected data. Therefore, it has O(1) complexity.

Variations:

None

2. Zooming



Steps to perform:

Position the cursor, then scroll the mouse wheel.

Procedural Steps:

OnceOrMore(MouseMove), OnceOrMore(Scroll)

Pseudo Code:

```
data[item].location.scale(cursor.location,count(Scr
ollEvent))
```

Application:

This gesture is used for scaling in and out data points on a plot.

Example:

Since the gesture's function is general, it can be used in many occasions regarding scaling in and out data points position.

Usability:

Since the cursor location is used as the pivot point, the user need make sure that the cursor location is correct before scrolling.

Performance complexity:

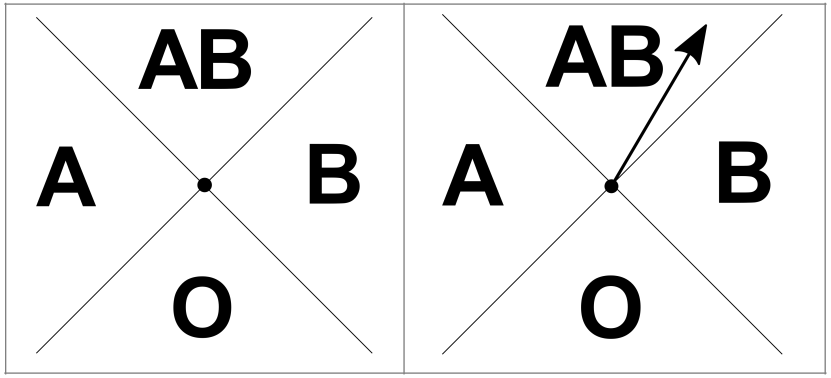
The user needs to click once for every single data. Therefore, it has O(N) complexity.

Variations:

None

Annotation Gestures

1. Two Bits Input



Steps to perform:

Mouse Click, then drag to a certain angle, release/second click to finish

Procedural Steps:

Once(MousePressed), OnceOrMore(MouseDragged), Once(MouseReleased)

Pseudo Code:

```
angle(point@1, point@2).map("AB", "A", "B", "O")
```

Application:

This gesture is used for giving an input for two bits information, such as blood type of a person.

Example:

Each combination of the two bits information can be mapped to any data that requires two independent variables, such as: turning on/off 2 lamps.

This gesture can be used in the case where a medical staff (e.g. nurse) needs to input the patient's blood type in a quick and concise way. Since it only needs a single swipe, it has the potential to process a batch of assigning blood type work in a short amount of time.

Usability:

The gesture is simple. The user only needs to click and swipe to a certain direction. The potential problem in using it is when the user swipe in the border area. Since the line is thin, the point correlates with the finger could be in the either side; thus, producing two different results.

One way to overcome this problem is by marking the area around the border as no-input area. But this solution can make the user repeatedly swipe to input the correct value. Another solution is by making the border line thicker. This solution can make the visualizations look ugly.

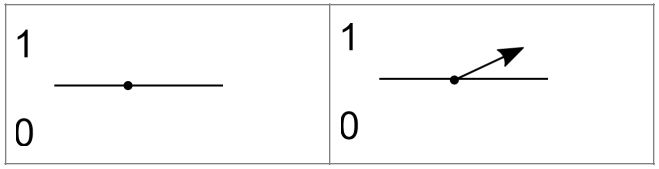
Special case(s):

The blood type can be represented as a double binary number, e.g. AB as 11, A as 10, B as 01, and O as 00. In this perspective, the gesture acts like an outlet for setting double on/off switches.

Performance complexity:

The blood type value is produced by making a single click and swipe. Since this gesture has a fix number of steps for achieving the goal, it has O(1) complexity.

2. Binary Input



Steps to perform:

Mouse Click, then drag to a up or down direction, release/ second click to finish

Procedural Steps:

Once(MouseClick), OnceOrMore(MouseDragged), Once(MouseReleased)

Pseudo Code:

```
point@1.horizontal.halfplanes.which(p@2).map("0","1
")
```

Application:

This gesture is used for giving a binary number as an input.

Example:

This gesture can be used in the case where students want to input binary numbers as part of their Math course. Since binary numbers can also be used to represent true/false or agree/disagree, it can be used to provide users a simple way to response to questionnaire questions and turning on/off switches.

Usability:

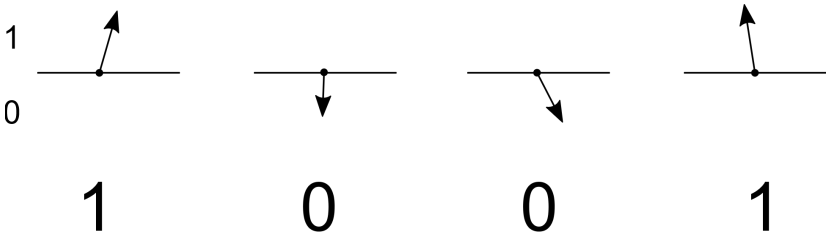
This is a simple gesture. The user only needs to click and swipe to a certain direction. The potential problem in using it is when the user's swipe is not long enough to distinguish the release location. In such case, the gesture is considered invalid and does not return any value.

Performance complexity:

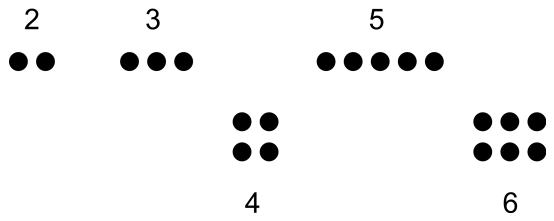
The binary value is produced by making a single click and swipe. Therefore, this gesture has a fix number of steps for achieving the goal (O(1)).

Variations:

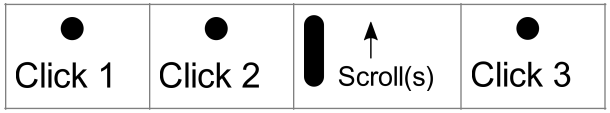
This gesture can be stringed together to provide the user with a mean to input many digit of binary number. It can also be used to be an on/ off switch.

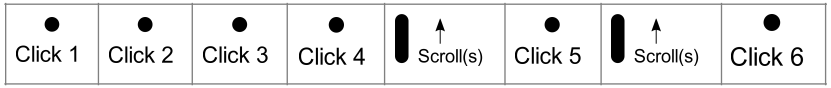


3. Prime and Composite Number Input



Prime: 1-dimensional, Composite: n-dimensional





Steps to perform:

Prime:

- 2 mouse clicks (a line), then mouse scroll. Each scroll corresponds to a predefined prime. number in increasing order (2, 3, 5, 7, 11, 13, 17, 19,23, 29, …). End with a mouse click.

Composite:

- 4 mouse clicks (2 lines), then mouse scroll to select an integer number. End with a mouse click. Repeat for the second time for the second number. The result is the multiplication result of the two numbers.
- 6 mouse clicks (3 lines), then mouse scroll to select an integer number. End with a mouse click. Repeat for the second time for the second number. Repeat for the third time for the third number. The result is the multiplication result of the three nu

Procedural Steps:

Once(MouseClick), Once(MouseClick), OnceOrMore(MouseScroll), Once(MouseClick)

Pseudo Code:

```
events.clicks.map(events.type(scroll).calculatePrim
e(), multiply(events.type(scroll).next(click),
events.type(scroll).next(click))
```

Application:

This gesture is used for giving a small prime or composite number as an input.

Example:

This gesture can be used in the case where biologists want to input small prime/composite number for recording the life cycle of periodic insect, e.g. Cicada (usually 7, 13, 17 years), Grasshopper, etc. This behavior is called Predator Satiation. It can also be used in the case where students want to input small prime numbers as part of their Math course.

Usability:

The first problem with this gesture is when the user lost count of the first click series, e.g. the user wants to input a 2-dimensional

composite number but provide click series for 3-dimensional composite number. To solve this, right-mouse-click can be used to signal that the user drops the previous click series.

The second issue is when the user wants to drop the input process whilst the user is already in the event process. The same as the previous solution, right-mouse-click can be used to signal that the user drops the series.

This gesture is only suitable to be used to input small prime or composite numbers, for example: the first 25 prime numbers between 1 and 100. If the numbers get bigger, the scrolling will get longer.

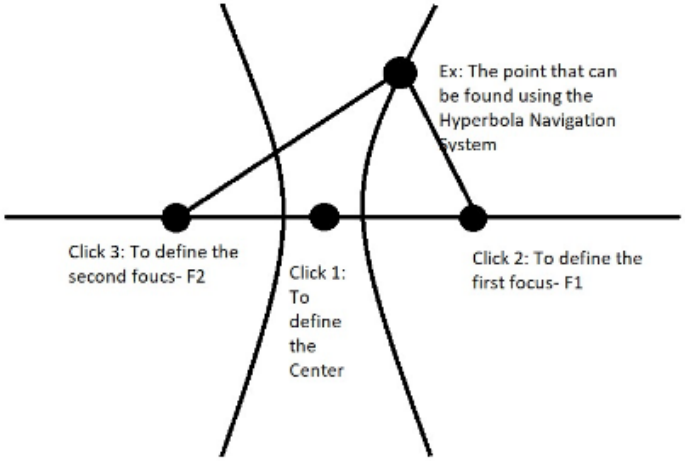
Performance complexity:

The prime and composite value is produced by making a series of mouse clicks and scrolls. For small numbers, the user needs to do up to 2N gestures, where N is the number being inputted. For larger numbers, the complexity decreases to log(N) gestures.

Variations:

None

4. Hyperbola Input



Steps to perform:

The first click will be used to define the center of the space. The second click will be used to define the first focus point. The third click will then define the second focus point of the hyperbola.

Procedural Steps:

Once(MouseClick), Once(MouseClick), Once(MouseClick)

Pseudo Code:

hyperbola(point@2 , point@3).linesegment(point@1)

Application:

The gesture can be used as the source that the user would like to find the location of by using the Hyperbolic Navigation System concept.

Example:

If two ships were at sea at different locations and both were transmitting signals above, if there were a plane above in the sky at a certain location such that the difference between the distance from the plane to the ships was always constant, then using this Hyperbolic Navigation System concept the precise location of the plane could be found. This position is such that it receives transmissions from both the ships at different rates.

Usability:

The user does not have to be able to click exactly on the hyperbola paths, clicking anywhere near the hyperbola paths defines the point that the user wants to find as the result of the application.

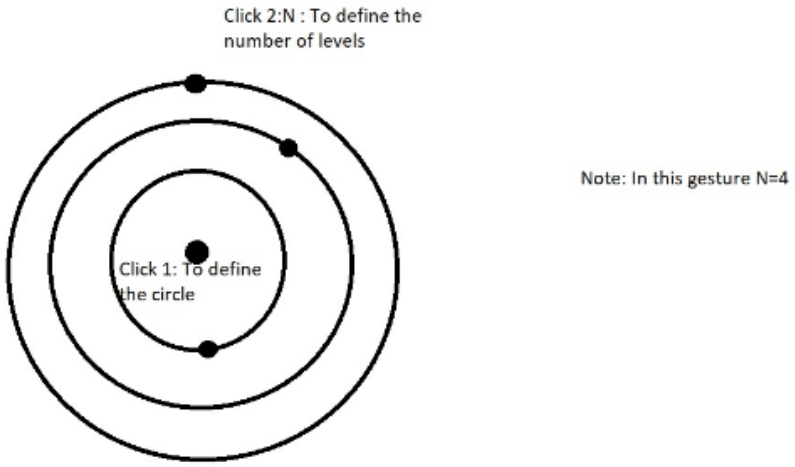
Performance Complexity:

Since this gesture has a fix number of steps for achieving the goal, it has O(1) complexity.

Variations:

None

5. Circle in circles Input



Steps to perform:

The first click will define the center of the circle. The second click and so on will define the number of circles the user wishes to draw so as to define the levels. Pressing shift button defines the end of the gesture.

Procedural Steps:

Once(MouseClick), OnceOrMore(MouseClick), Once(ShiftPressed)

Pseudo Code:

center(point@1).categoricalcircles(from(point@2..n))

Application:

This gesture can be used to interactively record the different levels of observations over a field or geographical data.

Example:

If the user has a device that would help them calculate the density ratio of a forest at different levels then, this gesture could help them visualize their information on, for example: an already existing map visualization of the forest. Using the clicks, the user can define their collected levels on the map to help them better understand the different boaters in the forest.

Usability:

The number of circles is given by the user to define the number of levels they would like.

The user will have to click the next set of clicks outside of the already existing circle for the gesture to make sense and work as expected.

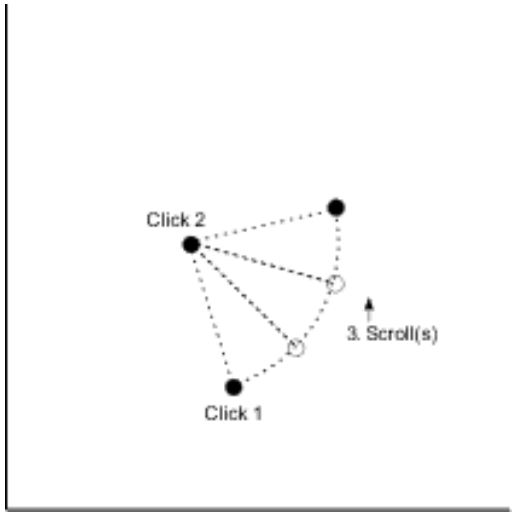
Performance Complexity:

The user needs to click once for every circle. Therefore, it has O(N) complexity.

Variation:

None

6. Scroll Rotation Annotation



Steps to perform

Click the first location/data, click the second location/center, scroll(s), click to finish

Procedural Steps:

Once(MouseClick), Once(MouseClick), OnceOrMore(MouseScroll), Once(MouseClick)

Pseudo Code:

```
rotate(point@2, point@1.location.select.data[item],
angle(scroll.events))
```

Application:

This gesture is used for rotating a data in a plot.

Example:

This gesture can be used in the case where the user needs to find an appropriate location for flash lights in a photography room. To get the best result, the flash lights needs to be put at 45 degree angle from the object and the camera. It can also be used to mark the location where corners should be when the user wants to build a circular fence around a building. Or it can also be used simply to rotate a data point in a geometry based plot.

Usability:

The issues with this gesture are to accurately select the center of rotation. Getting it off by only a little bit can produce a big difference for the data's final position.

The rotation follows the rotation of the mouse wheel; i.e. clockwise for down scroll and counter-clockwise for up scroll. For scroll-ball-based mouse, the rotation correspond directly with the angle of the object rotation. For wheel-based-mouse, each scroll corresponds with 15 degree rotation.

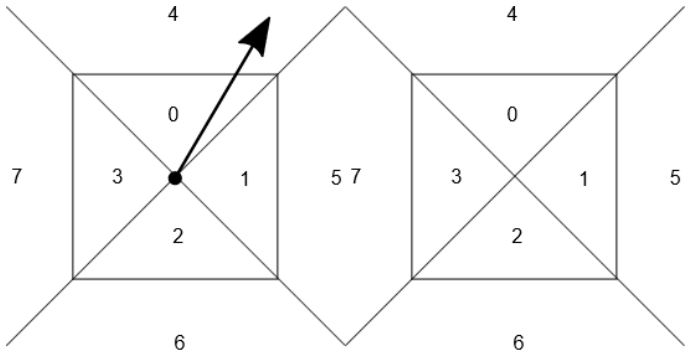
Performance complexity:

Every single rotation needs three clicks and several scrolls. Therefore, this gesture has O(N) for its performance complexity.

Variations:

The data point to be rotated can be of any object, e.g. building on map, equipment in factory, etc.

7. Hex Number Input



Steps to perform:

Mouse Click, then drag to a certain angle and length, release/ second click to finish

Procedural Steps:

Once(MousePressed), OnceOrMore(MouseDragged), Once(MouseReleased)

Pseudo Code:

```
point@1.angle(point@2).length(point@2).map(0, 1, 2, 3, 4, 5, 6, 7)
```

Application:

This gesture is used for inputting a single digit number from 0 to 7.

Example:

This gesture can be used in the case where computer scientists want to input a hex number as part of his programming task. It can also be used in the case where students want to input hex numbers as part of their Math course.

Usability:

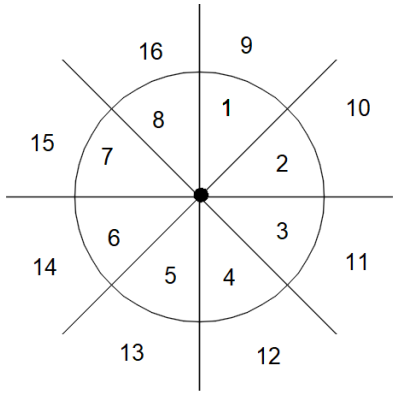
To use the gesture, user needs to click and drag to a certain direction and certain length. The issue is that the users need to learn how much they should drag to produce the inner or outer numbers.

Performance complexity:

The hex number is produced by making a single click and swipe. Therefore, this gesture has a fix number of steps for achieving the goal.

Variations:

This gesture can be extended to be a tool capable to input a set of number starting from -7 to 7 by using the second mouse button, i.e. the left mouse button is used to produce 0 to 7 number and the right mouse button is used to produce 0 to -7 number. The layout of the number is still the same; the difference is only that instead of producing positive number, the second button produces negative numbers. It can also be modified to be able to input a set of number (1 to 16) by using eight divider lines and also using circle instead of a square.



8. Ratio Input

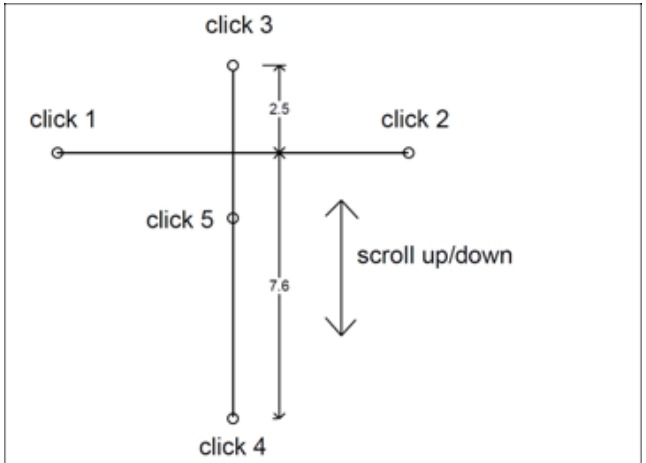


Figure 1: Gesture to specify ratio of line segment above to below the horizontal line

Steps to perform:

Click any two points (click1 and click 2) in the space to define a reference line segment. Click two points (click3 and click4) on the plane to form a vertical line segment intersecting the line segment drawn in step 1. Click any point (click5) on the vertical line drawn in step 2 and scroll the mouse up/down to adjust the length of line segments above and below the line drawn in step 1 to represent the ratio of length of line segment above the reference line to length of line segment below the reference line.

Procedural steps:

Once(MouseClick), Once(MouseClick), Once(MouseClick), Once(MouseClick), Once(MouseClick), OnceOrMore(MouseScroll)

Pseudocode:

```
ratio(length < geom.line(click1, click2) ,
click@3(geom.line(click@3,click@4)>,
length<geom.line( click1,click2),click4(geom.line(c
lick@4,click@5))>).scroll(translate.along.click@5.l
ine)
```

Application:

The gesture is used to enter ratios. Figure1 represents the gesture without round off to enter approximate ratio and adjust it to precise value using the mouse scroll .

Example:

A civil surveyor might use this gesture to specify the ratio of height of the building to the tallest building allowed to construct in a certain soil according to safety standards. By providing a grid layout under the background the user will be able to count the grids and define the ratio in step 2 without the need to scroll the mouse.

This gesture can be used to specify the ratio of apples to oranges grown in an area in tons. It can also be used to define the ratio of supporters of one political party to the other in a demographic area. It can be used to dine the ratio of international students to the others in a college.

Usability:

The first usability issue for the user is he should select the points horizontally or vertically. i.e the points should be on the same plane. if they don't lie on the same plane. then the projection of the hover

point should project on to the horizontal or vertical plane to assist the user in selecting the point.

The user might select the vertical line not bisecting the horizontal line. in that case the system should not allow the user to draw the vertical line that is not bisecting the horizontal line.

Performance Complexity

It takes constant time to define the horizontal and vertical lines, but it takes linear time to adjust the vertical line. Therefore, the complexity is O(N).

Variations:

The variation of this gesture is shown in figure 2 which allows the users to enter ratios with roundoff.

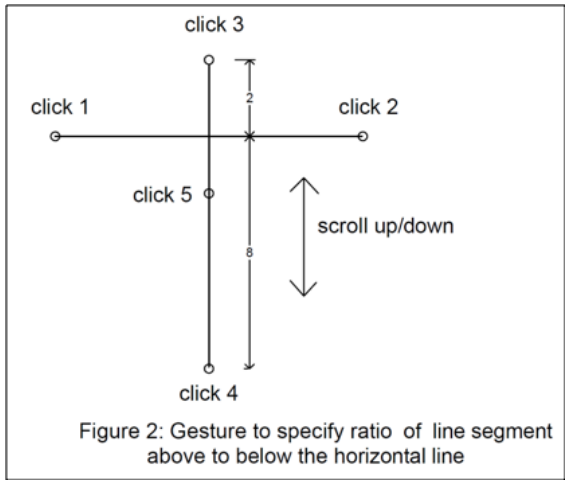


Figure 2: Gesture to specify ratio of line segment above to below the horizontal line

9. Multidimensional Value Input

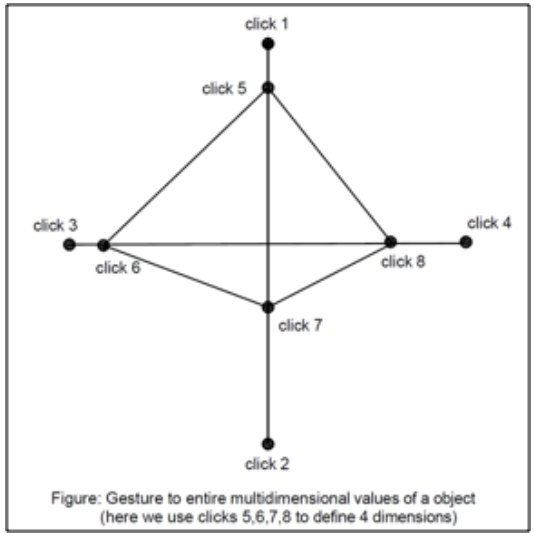


Figure: Gesture to entire multidimensional values of a object
(here we use clicks 5,6,7,8 to define 4 dimensions)

Steps to perform:

Click any two points (click1 and click 2) in the space to define a horizontal line segment. Click two points (click3 and click4) on the plane to form a vertical line segment intersecting the line segment drawn in step 1. By defining a scale and dimension for each axis which can be categorical, numerical or ordinal. The user will be able to select different points o the axis and draw a star plot.

Procedural Steps:

Once(MouseClick), Once(MouseClick), Once(MouseClick), Once(MouseClick), Once(MouseClick), Once(MouseClick), Once(MouseClick), Once(MouseClick), Once(MouseClick),

Pseudo Code:

geom.axis(clicks@1)(click@2,click@3,click@4).length(intersection(axis), clicks)

dimension_line12 = glyph.line(click1, click2)

dimension_line34 = glyph.line.intersect.dimension_line12(click3, click4)

dimension_line12.first.value = distance(click5, intersection(dimension_line12, dimension_line34)

dimension_line12.second.value = distance(click7, intersection(dimension_line12, dimension_line34)

dimension_line34.first.value = distance(click6, intersection(dimension_line12, dimension_line34)

dimension_line34.second.value = distance(click8, intersection(dimension_line12, dimension_line34)

Application:

The gesture defined in the above diagrams is used to enter multi-dimensional values. Figure1 represents the gesture which allows the user to enter four different kinds of values for any given object using the axis of a star plot. The origin represents a value of zero when entering numerical values with increasing towards the end of axis. This can be used to enter direct values or calculate derived values from different dimensions represented on the star plot such as ratio of any two dimensions that are numerical.

Example:

This is a general-purpose gesture that can be used to values of various dimensions. A maintenance person might use this gesture to track the different variables for an asset like a building by using first axis to enter the height of the building, second axis to enter the width of the building, third axis to enter the number of rooms in the

building and fourth axis to represent the color of the building. The gesture would save direct values like height of the building or derived values like height times width of the building

This gesture can also be used to enter the condition of an archeological specimen on one axis, the age of the specimen on another axis, the category of the specimen on another axis and the number of similar items found in the area on another axis.

Usability:

This gesture can be used to enter data values of different types such as numerical, categorical, ordinal. care should be taken such that categorical values do not represent any ordinal values. If the author tries to enter odd number of dimensions he still has to choose even number of axis as a frame of reference and default one of the dimension to zero.

Performance Complexity:

It takes linear time to define the axis because he has to count the dimensions he is representing while defining the axis. It also takes linear times to enter the data as he has to count the distance on the axis while he is entering the data. Therefore, it has O(N) complexity.

Variations:

None

10. Perpendicular distance input

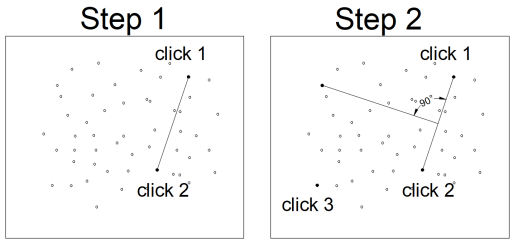


Figure: Gesture to define perpendicular distance between point and a line segment. step1:Define the line(click1,click2) step2:Select the point(click3)

Steps to perform:

Click two points(click1) and (click2) in the space to define a reference line. Click a third point (click3) on the plane for which you need the perpendicular distance from the reference line.

Procedural Steps:

Once(MouseClick), Once(MouseClick), Once(MouseClick)

Pseudo Code:

ReferenceLine = Glyph.line(click1, click2)

Restricted_plane = ReferenceLine.project(axis, 90)

ReferencePoint = glyph.point(Restricted_plane.select(click3))

if(exists(ReferencePoint){

perpendicular_distance = distance.perpendicular(ReferencePoint, ReferenceLine)

}else{

closest distance = distance(click3, ReferenceLine)

}

Glyph.line(click1, click2).perpendicularLine(point@3).length

Application:

The gesture can be used to define the scenarios where we need the perpendicular distance between the point and the reference line.

Example:

This is a gesture that can be used to select perpendicular distance between the selected point and a reference line which can be road or a river flowing in a straight line.

Usability:

For any point selected by click 3 if it does not lie in the plane perpendicular to the line formed by click 1 and click 2 it won't be highlighted or allowed.

If the user tries to select a point outside the plane, he cannot see the feedback given by the visualization which is a perpendicular line instead it would show the closest distance between the reference point and the reference line by using a non perpendicular line.

Performance complexity:

Since it has a fixed number of operations to reach the goal, it has O(1) complexity.

Variations:

Variation 2

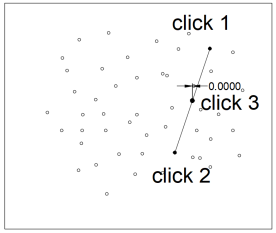


Figure: Point lies on the line.

Variation 3

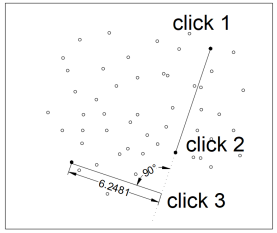


Figure: Point lies on perpendicular to the extension of the line.

11. Closest distance Input

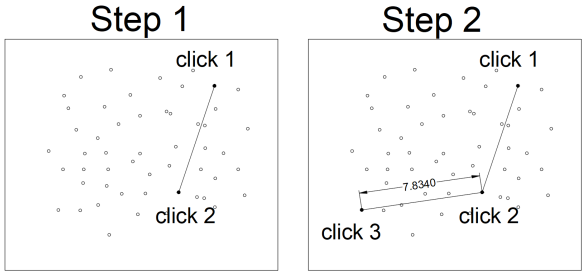


Figure: Gesture to define closest distance between point and a line.
step1:Define the line(click1,click2) step2:Select the point(click3)

Steps to perform:

Click any point (click1) in the space to define a reference point. Scroll the mouse to define the area of the circle with the first selected point in step 1 as its center.

Procedural Steps:

Once(MouseClick), Once(MouseClick), OnceOrMore(MouseScroll)

Pseudo Code:

ReferenceLine = Glyph.line(click1, click2)

ReferencePoint = glyph.point(click3)

closest distance = distance(click3, ReferenceLine)

Application:

This gesture is used to find the shortest between a point and a line. This is very similar to perpendicular distance but the difference is when the point does not lie on the perpendicular to the reference line then a slanted line is drawn to connect the point and the line.

Example:

This gesture can be used for finding the shortest distance to a road segment or mass transport path.

Usability:

For any point selected by click 3 if it lies outside plane perpendicular to the line formed by click 1 and click 2 , then a straight line would be drawn from the point to the closest end of line segment.

Performance Complexity:

Since it has a fixed number of operations to reach the goal, it has O(1) complexity.

Variations:

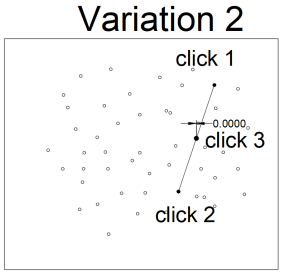


Figure: Point lies on the line.

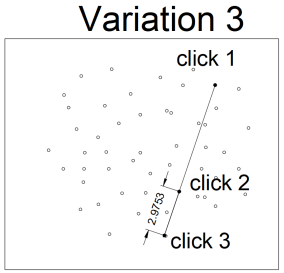


Figure: Point lies on the extension of the line.

12. Angle Based Slope Input

Steps:

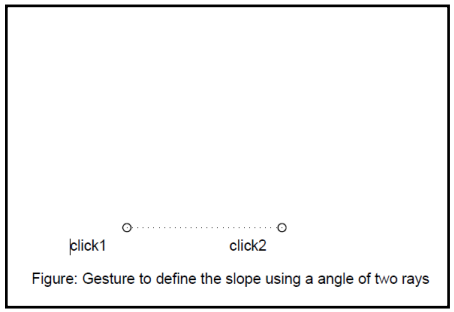


Figure: Gesture to define the slope using a angle of two rays

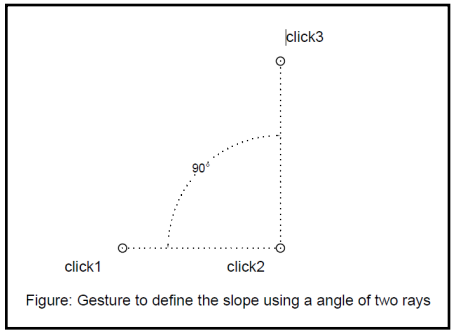


Figure: Gesture to define the slope using a angle of two rays

Steps to perform:

Click any two points (click1 and click 2) in the space to define a line segment. Click two points (click3 and click4) on the plane to form a line segment intersecting the line segment drawn in step 1. Click any point (click5) in the space to select an area that falls on any of the four sides that are formed by the intersecting lines

Procedural Steps:

Once(MouseClick), Once(MouseClick), Once(MouseClick), Once(MouseClick), Once(MouseClick)

Pseudo Code:

Line Reference_line1 = glyph.line(click1, click2)

Line Reference_line2 = glyph.line(click2, click3)

angle Slope = angle(Reference_line1, Reference_line2)

Application:

This gesture is used to define the slope using three points. The slope always ends up between (0 and 180). It takes the first two points to draw the reference line and then used the third point to define an angle from the second point.

Example:

This gesture can be used to define the slope of the road, slope of the mountain.

Usability:

One of the difficulties associated with this gesture is the user might want to define the angle formed by click2, click1, and click 3 taken in order rather than click1, click2, click3 taken in order.

The other problems are the user might use the same point either click1 or click2 to define click3 which will not result in an angle.

Performance Complexity:

Since it has a fixed number of operations to reach the goal, it has O(1) complexity.

Variations:

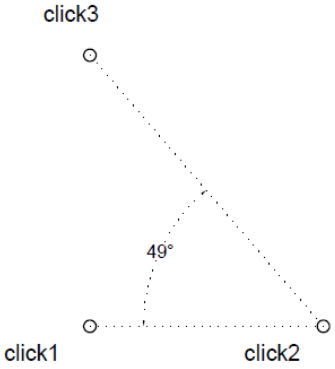


Figure: Gesture to define the slope using a angle of two rays

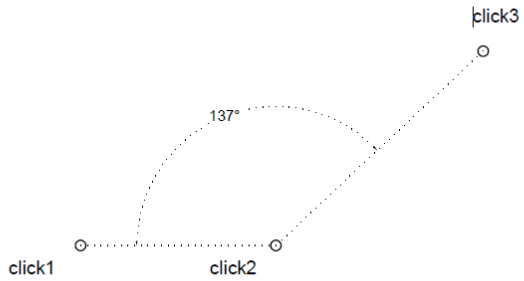


Figure: Gesture to define the slope using a angle of two rays

13. Sine curve cycles Input

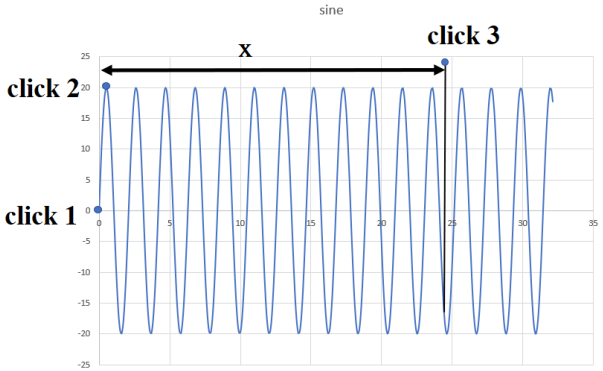


Figure: Gesture to enter large numbers by measuring the length of the sine wave

Steps to perform:

Click a point(click@1) to define the point at which the sine curve touches the mid line to define the beginning at phase 0. Click a point (click@2) to define the amplitude of the sine curve at quarter cycle i.e. at phase 90 degrees. Click a point anywhere above the sine curve on the plane (click@3) to draw a perpendicular line on to the x axis that defines that touches the sine curve. The number of complete cycles from click@1 to this intersecting point defines the number.

Procedural Steps:

Once(MouseClick), Once(MouseClick), Once(MouseClick)

PseudoCode:

point phase0_point = geom.point(Click@1)

length amplitude = distance(click@2, click@1.horizontal)

curve sine_curve = geom.sine(phase0_point, amplitude)

data[selected].item = sine_curve.cycles(click@1, intersection(click@3.projection(sine_curve)))

Application:

The sine curve can be used to enter values related to a cyclic phenomena. By selecting any point of the plane above the sine curve, the user can the count of complete cycles from the from the first selected point to the intersection of vertical to the sine curve from the last selected point.

Example:

This can be used to enter the cyclic values like revolutions per minute of the vehicle and a satellite rotations.

Usability:

Since the sine wave size is fixed, this gesture is difficult to use for inputting a large number of cycles.

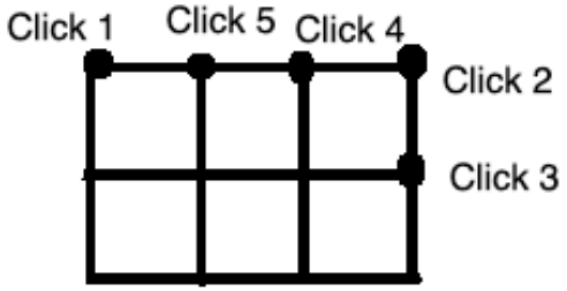
Performance Complexity:

Since it has a fixed number of operations to reach the goal, it has O(1) complexity.

Variations:

None

14. Square-Grid Input



Steps to perform:

Click 1 to define the starting of the side of a square. Click 2 to define the end of the side of the square. The next set of clicks will define the partitioning of the square.

Procedural Steps:

Once(MouseClick), Once(MouseClick), NoneOrMore(MouseClick)

Pseudo Code:

square(point@1,point@2).grid(point@3..n)

Application:

This gesture can be used to visually display divided data or divide the data visually and edit it using the interactions provided.

Example:

Using this gesture we can interactively define and edit any community structure plans. For example, the defining of the parts where the houses are built could be done using the clicks and to change the dimensions of these different parts we could use the drag gesture.

Another example is to use this gesture to visually encode different types of grids such as Modular or Hierarchical grids type data such as the placement of icons in our phones and use the interaction provided to visually edit the data being displayed by using the click drag release gesture.

Usability:

A click drag release option can also be provided for interactively changing the dimensions of the squares. In such case, if the users click the interaction point at a distance equal from two or more sides, then it will be difficult for the gesture to decide which side should it change.

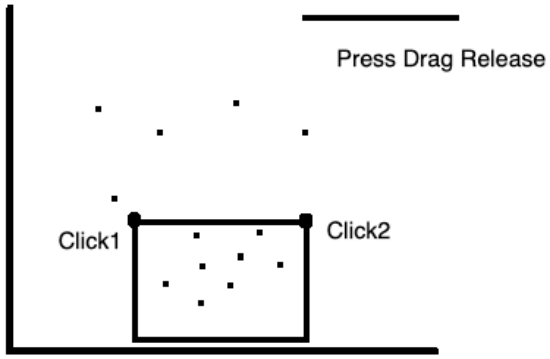
Performance Complexity:

Since it has a set of clicks similar to the number of square divides, it has O(N) complexity.

Variations:

None

15. Square - Selection and Interaction



Steps to perform:

Click 1 to define the starting of the side of a square. Click 2 to define the end of the side of the square. Press shift and a click drag release outside of the square to define the unit of charge for to be applied on the selected data points.

Procedural Steps:

Once(MouseClick), Once(MouseClick), Once(MousePressed), OnceOrMore(MouseMove), Once(MouseReleased)

Pseudo Code:

square(point@1,point@2).click.drag.release()

Application:

This gesture can be used to edit data values after selecting them using a square.

Example:

Let us assume we have a scatter plot. Using this gesture we can select a set of points on the plot and edit all of these values at the same time.

Usability:

A click drag release to interactively edit data values with in the selected region. As an example of the editing that can happen here is changing the axis values of the data.

To distinguish this gesture with data moving gesture, the user cannot perform the press drag release inside the square.

Performance Complexity:

The square selection has constant complexity, whereas the editing might need many dragging operations. Thus, in total, the gesture complexity is O(N).

Variations:

None

16. Hexagon-Special Case Input

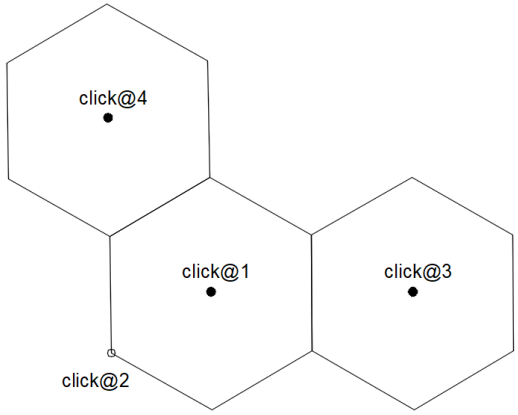


Figure : Hexagon Gesture Using two clicks

Steps to perform:

The first click defines the center of the hexagon. The second click is the end of the gesture that defines the length of the side. The next set of clicks define the number of hexagons and the position the user would like attached to the existing hexagon.

Procedural Steps:

Once(MouseClick), Once(MouseClick), OnceOrMore(MouseClick)

Pseudo Code:

Hexagon(point@1, point@2).center(point@1).position(point@3...n)

Application:

This gesture is for creating a tight knit hexagons

Example:

The gesture can be used to select all the optimal positions for antenna placement within an area.

Usability:

By default, the hexagons produced will be aligned perpendicular to the axis. No matter where the second click performed, the hexagon will always be perpendicular, i.e. the second click does not always represent a vertex. It can be a part of the side line.

The user should make sure that the side of the hexagon is not as too big for the provide space while keeping in mind the total number of hexagons they would like to have in the figure.

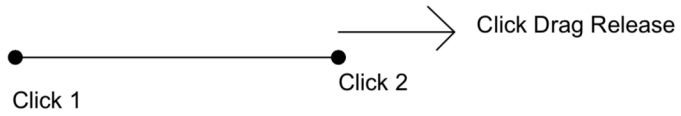
Performance Complexity:

The gesture needs a number clicks similar to the number of hexagons. Thus, in total, the gesture complexity is O(N).

Variations:

If the user wants to create rotated hexagons, the gesture can be modified such that the second click always defines a vertex.

17. Line Segment - Annotation



Steps to perform:

The first click will define the starting point of the line segment. The second click will define the end point of the line segment. The next step is the interaction where the user can visually edit the data being represented by the line segment by clicking on any of the end points and dragging the line segment.

Procedural Steps:

Once(MouseClick), Once(MouseClick), Once(MousePressed), OnceOrMore(MouseMove), Once(MouseReleased)

Pseudo Code:

linesegment(point@1, point@2).edit(point@1,point@2).length.map(designerValues)

Application:

This gesture can be used to visually edit data being represented in the form of a line segment.

Example:

Let us assume the count of the number of participants in an event is being represented using this gesture. By using the interaction provided, the user can visually decrease or increase this count say from 200 to 150 or 200 to 300 respectively.

Usability:

The interaction provided in this gesture is a click drag release on any of the two end points. This will allow the user to visually edit the data values that is being represented by the line segment.

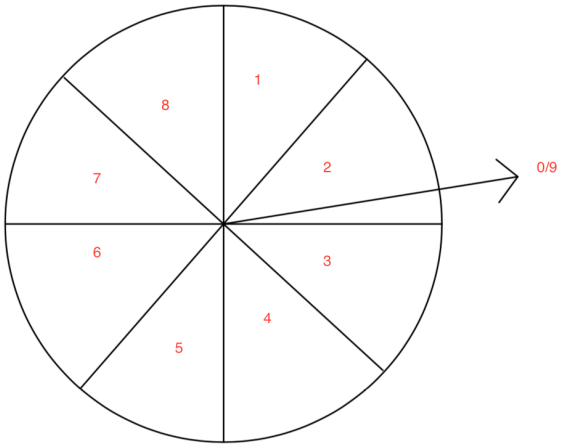
Performance Complexity:

The line segment creation has constant complexity, whereas the editing might need many dragging operations. Thus, in total, the gesture complexity is O(N).

Variations:

None

18. Circle- Single Digit Integer Input



Steps to perform:

Firstly, the click-drag-release defines the radius of the circle. At this step the circle is automatically divided in to 8 parts. The next set of clicks will define the number that the user would like to input. The center of the circle represents 0 and 9. A right click represents 9 and a left click represents 0.

Procedural Steps:

Once(MousePressed), OnceOrMore(MouseMove), Once(MouseReleased), Once(MouseClick)

Pseudo Code:

Circle(point@1,point@2).selection(point@3 ...11)

Application:

This gesture can be used to input single digit integers.

Example:

This gesture can be used in cases where the user would like to input any categorical values up to 10 categories.

Usability:

For easy to use, a visual feedback might be provided for the user showing which parts of the circle represent which number.

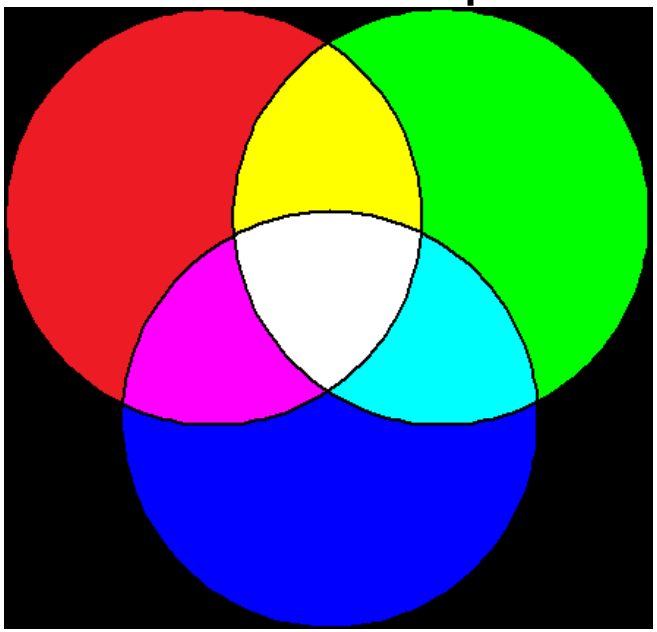
Performance Complexity:

Since it has a fixed number of operations to reach the goal, it has O(1) complexity.

Variations:

None

19. RGB Color Input



Steps to perform:

Mouse Click, then drag to a certain angle and length, release to finish

Procedural Steps:

Once(MousePressed), NoneOrMore(MouseDragged), Once(MouseReleased)

Pseudo Code:

```
p@2.inside(redCircle, greenCircle,
blueCircle).map("white", "yellow", "magenta",
"cyan", "red", "green", "blue", "black")
```

Application:

This gesture is used for inputting a primary RGB color.

Example:

This gesture can be used in the case where designers want to input a color as a part of their creative work. It can also be used in any other cases where users want to input color value as a part of their tasks.

Usability:

To use the gesture, user needs to click and drag to a certain direction and certain length. The issue is that the users need to learn how much they should drag to produce the desired value.

Performance complexity:

The color value is produced by making a single click and swipe. Therefore, this gesture has a fix number of steps for achieving the goal (O(1)).

Variations:

The position of Black and White color can be swapped in certain case where the users need to input black value more frequently.

20. Square Grid Integer Input

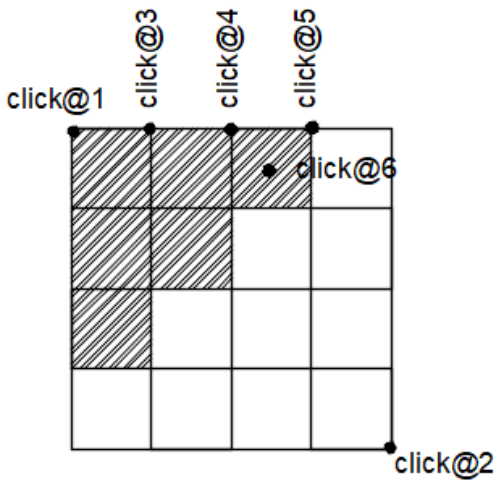


Figure: Brightness using diagonal fill

Steps to perform:

Click a point (click@1) to represent the start of the NxN square grid. Click a point (click@2) to represent the end of the NxN square grid. Select n-1 points on any side of the NxN grid to represent the NxN square grid. Select any cell in the grid to fill the grid from top left corner diagonally filling the cells left to right across diagonal.

Procedural Steps:

Once(MouseClick), Once(MouseClick), OnceOrMore(MouseClick), Once(MouseClick)

Pseudo Code:

```
square square = geom.square(click@1, click@2)

GridSquare = square.grid(click@3,click@4,Click@5)

data[item].selected
=count( GridSquare.DiagonalFill(click@5))
```

Application:

This is gesture can be used to select an even integers from a group of integers. This is a modification to slider input with the additional advantage of grouping the range into sub groups and giving the user ability to reach closer to 50% of the data more easily than the slider input.

Examples:

This can be used to enter brightness of the displays which is normally a 16x1 grid on computers. Users can easily navigate close to 50% brightness by visual cue of the symmetry of the square.

Another example of this gesture is use a 10x10 grid and multiply the count the filled rectangles with 0.1 and add to the base body temperature of 96 to get a temperature ranges of 96 to 106.

Usability:

One of the difficulties associated with this gesture is if the user does not select the points after the first two points on a same side, it forms a rectangular grid instead of square grid. Since the filling of the square is non-linear in nature when moved along the first row, i.e (1, 3, 6,10,13,15,16). This form of filling the grids has non-linear nature at the beginning and the ends of the brightness scale and easy to enter values close to half of the range.

Complexity:

Defining the initial setup for the gesture takes order of N/2 for a NxN grid. And selecting a grid takes a non linear time for values between 0 and 50%, 50% and 100%. It takes constant time for 0%, 50%, and 100%. Therefore, it has O(N) complexity.

Variations:

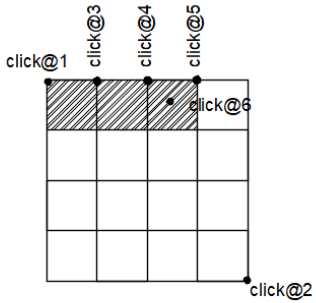


Figure: Brightness using rectangular fill

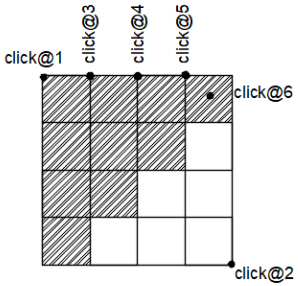


Figure: Brightness using diagonal fill

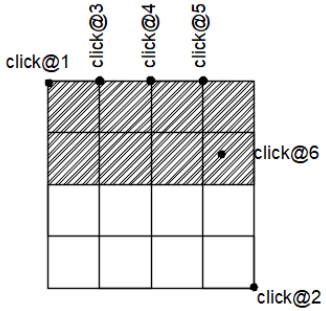


Figure: Brightness using rectangular fill

21. Decimal Input

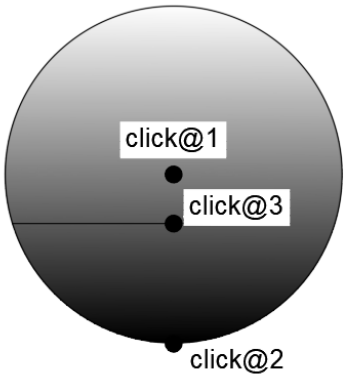


Figure:Gesture to represent Brightness

Steps to perform:

Click a point(click@1) to define the center of the circle. Click a point (click@2) to define the radius of the circle. Click a point (click@3) to define the brightness level.

Procedural Code:

Once(MouseClick), Once(MouseClick), Once(MouseClick)

Pseudo Code:

GradientCircle circle = geom.circle(click@1, click@2).gradient(black, white)

data[selected].item = circle.gradient(click@3)

Application:

The circle can be used to select a value that follows the properties of a Color Gradient, which is usually difficult to calculate without the use of specialized sensor.

Example:

The circle can be used to define the brightness of the enclosed spaces or an outdoor location.

A photographer might use this gesture to define the brightness of a photo shoot location. A weather reporter might use this to define the brightness of the day.

By defining the brightness of the day using direct color gradient from white to black, the user can precisely note the brightness or darkness of the room. Since brightness and darkness complement each other, this gesture can be used to calculate both of them.

Usability:

One of the usability issue associated with this gesture is the user might think the brightness also varies horizontally which is not the case. We can use the circular gradient as an alternative.

Performance complexity:

It takes constant time (O(1))to define the gesture as well as perform the operation.

Variations:

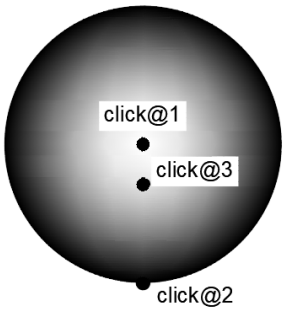
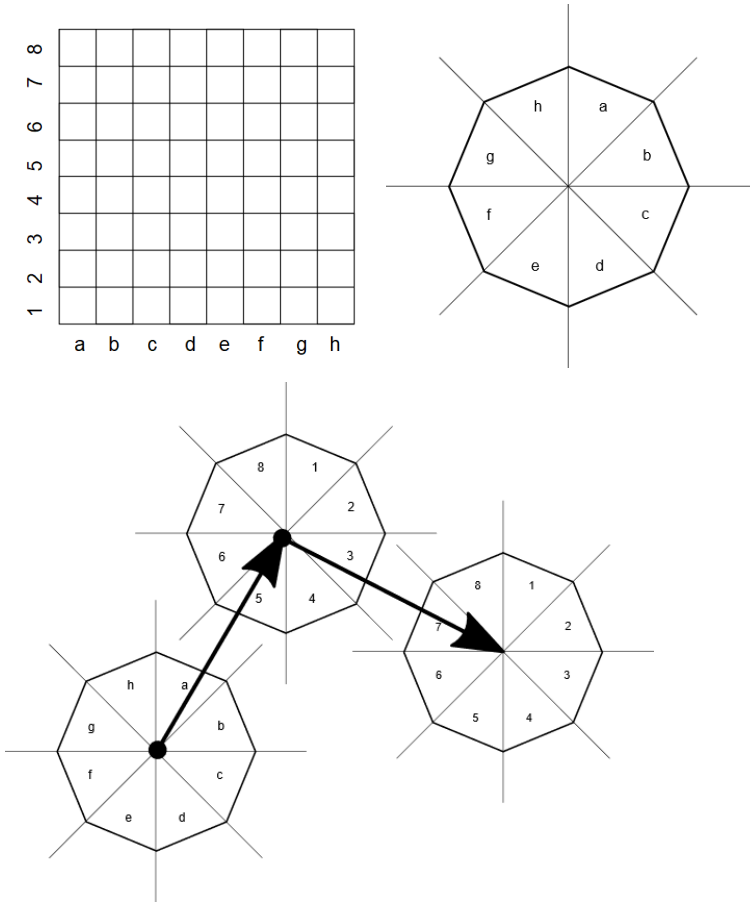


Figure:Gesture to represent Brightness

Figure:Gesture to represent Brightness

22. Grid Data Selection and Input



Steps to perform:

For specifying grid size:

- Users can use a variant of "Hex Number Input" Gesture: mouse click, then drag to a certain angle and length, release/second click to finish; or "Square-Grid" Gesture: Two clicks to define the square. The subsequent clicks enable the user to divide the square however he wishes.

For selecting a cell:

- Mouse Click; then drag to a certain angle; after the drag reaches certain length, the column value corresponding to that angle is selected; drag to a certain angle and length again to select row value; release/second click to finish

For inputting the data:

- Use "Hex Number Input" Gesture: mouse click, then drag to a certain angle and length, release/second click to finish

Procedural steps:

Once(MousePressed), OnceOrMore(MouseDragged), Once(MouseReleased), Once(MousePressed), OnceOrMore(MouseDragged), OnceOrMore(MouseDragged), Once(MouseReleased), Once(MousePressed), OnceOrMore(MouseDragged), Once(MouseReleased)

Pseudo Code:

```
p@1.angle(p@2).length(p@2).map(a, b, c, d, e, f, g,
h).concatenate.p@2.angle(p@3).length(p@3).map(1, 2,
3, 4, 5, 6, 7, 8).concatenate.p@3.angle(p@4).length
(p@4). map(1, 2, 3, 4, 5, 6, 7, 8)
```

Application:

This gesture is used for selecting and setting a value to all data points in the selected cell.

Example:

This gesture can be used in the case where data scientists want to assign a group of dataset to a certain category. It can also be used when city planners want to assign a group of neighborhood to a certain set of category.

Usability:

To use the gesture, user needs to click and drag to a certain direction and certain length thrice. The issue is that the users need to learn how much they should drag to produce the desired value.

The maximum number of column and row is 8. The reason for this is the gesture will be difficult to use if there are more than 8 angles for giving the input value.

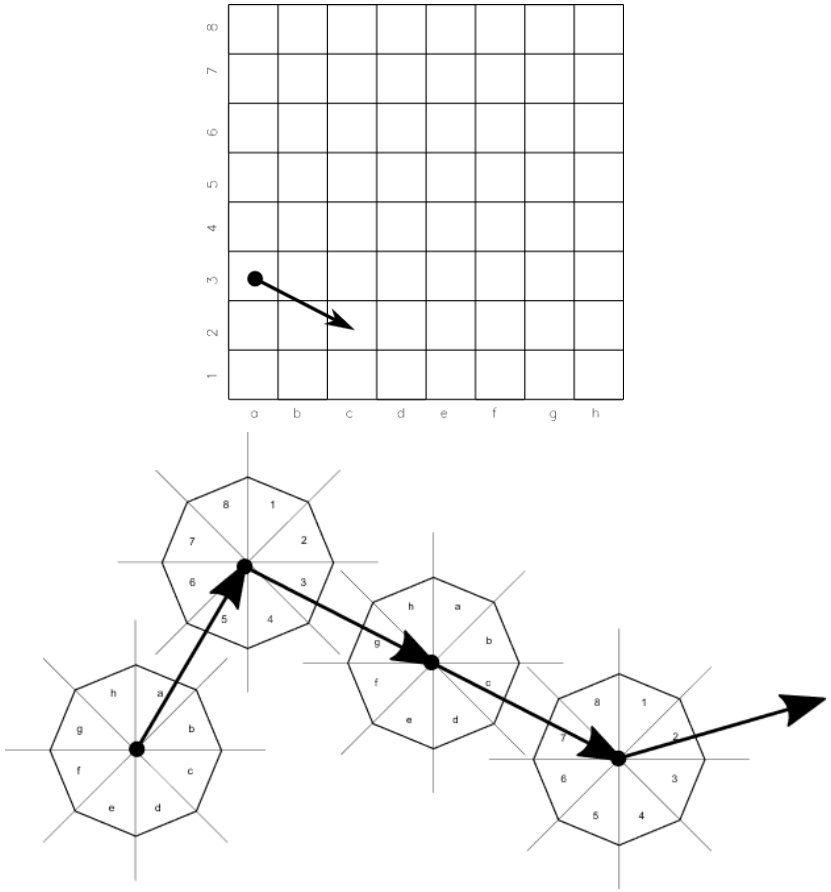
Performance complexity:

The selection and input processes are done by making click and drag gesture thrice. The dragging can be of many instances. Therefore, this gesture has O(N) complexity.

Variations:

The grid shape does not have to be square. As long as it can be indexed from a to h and 1 to 8, it can be used. For example, we can use "Hexagon-Special Case" gesture to create the grid; in which case, we use hexagon as the grid's cell.

23. Movement in Grid Input



Steps to perform:

For specifying grid size:

- Users can use "Hex Number Input" Gesture: mouse click, then drag to a certain angle and length, release/second click to finish; or "Square-Grid" Gesture: Two clicks to define the square. The subsequent clicks enable the user to divide the square however he wishes.

For inputting a movement:

- Mouse Click; then drag to a certain angle; after the drag reaches certain length, the column value corresponding to that angle is selected; drag to a certain angle and length again to select row value. This will select the origin cell. Repeat to select the destination cell. Release/second click to finish

Procedural steps:

Once(MousePressed), OnceOrMore(MouseDragged), OnceOrMore(MouseDragged), OnceOrMore(MouseDragged), OnceOrMore(MouseDragged), Once(MouseReleased)

Pseudo Code:

```
p@1.angle(p@2).length(p@2).map(a, b, c, d, e, f, g, h).concatenate.p@2.angle(p@3).length(p@3).map(1, 2, 3, 4, 5, 6, 7, 8)p@3.angle(p@4).length(p@4).map(a, b, c, d, e, f, g, h).concatenate.p@4.angle(p@5).length(p@5).map(1, 2, 3, 4, 5, 6, 7, 8)
```

Application:

This gesture is used for inputting a movement from an origin cell to a destination cell.

Example:

This gesture can be used in the case movement is constrained to a gridded world, for example: chess, robotic movement, character movement in strategy games, and certain war/transportation simulation.

Usability:

To use the gesture, user needs to click and drag to a certain direction and certain length four times. The issue is that the users need to learn how much they should drag to produce the desired value.

The maximum number of column and row is 8. The reason for this is the gesture will be hard to use if there are more than 8 angles for giving the input value.

Performance complexity:

The movement input process is done by making click and drag gesture four times. The dragging can be of many instances. Therefore, this gesture has O(N) complexity.

Variations:

The grid shape does not have to be square. As long as it can be indexed to a to h and 1 to 8, it can be used. For example, we can use "Hexagon-Special Case" gesture to create the grid; in which case, we use hexagon as the grid's cell.

24. Ratio Input Using Rectangle

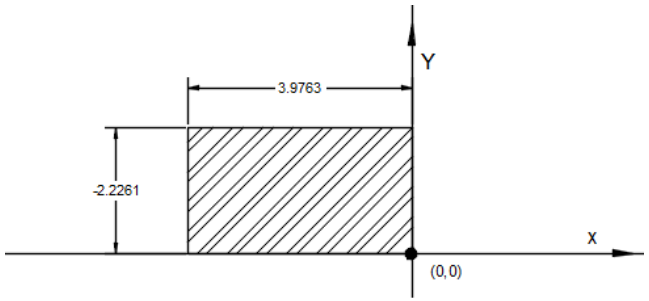


Figure: Gesture to define the ratio using (x,y) coordinates of Rectangle

Steps to perform:

Click to define the origin and then click anywhere to form a rectangle with the second click defining the diagonal of the rectangle.

Procedural steps:

Once(MouseClick), Once(MouseClick)

Pseudocode:

ratio(abscissa(click@1),ordinate(click@1))

Application:

This gesture is used to show ratios using the length and width of the rectangle. By defining the origin as one of the vertices of the rectangle we can get positive and negative ratios.

Example:

A civil surveyor would like to note the ratio of length by width of land plots from the reference point.

An architect might use this gesture to study the buildings.

This gesture can be used to define the aspect ratio of screens of computers, tablets and phones.

Usability:

To define the ratio, user needs to click at any point on the space define the first vertex of the rectangle on the bottom side and click a second point to form to define the opposite vertex of the rectangle. Here we are using the XY plane as the reference and the first click as the origin.

By using the XY coordinate system as the reference any rectangle drawn falls in one of the four quadrants. As a result we can get negative ratios and positive ratios based on the abscissa and ordinates of the second click.

This gesture is suited to specify ratio between two different quantities rather than the ratio a part to whole.

Performance Complexity:

The gesture takes nonlinear time to select create a rectangle as per the required dimensions of length and width.

Variations:

None

25. Ratio Input Using Concentric Circles

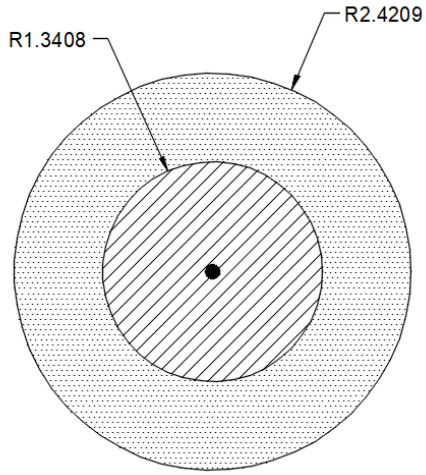


Figure: Gesture to enter ratio using area of circles

Steps to perform:

Click to define the centre of the circle and then click again to define the radius of the inner circle. Next, click anywhere to form the outer circle to define the whole.

Procedural steps:

Once(MouseClick), Once(MouseClick), Once(MouseClick)

Pseudocode:

ratio(area(circle(click@1, click@3)) , area(circle(click@1,click@2)))

Application:

This gesture is used to define ratios using the area or two circles with same center. The area of the outer circle represents the whole quantity and the area of the inner circle represents the partial quantity out of whole.

Example:

An forest surveyor could use this gesture to define the ratio of a particular vegetation in a forest. Here, he usually defines the ratio by eyeballing the forest.

A Forest department might also use this gesture to represent the ratio of destroyed forests by some phenomena like forest fires which usually spread radially.

Usability:

To define the ratio, user needs to click at any point on the space define the centre of both the circle. He then clicks on the plane to represent the outer circle to represents the whole of any quantity and then makes a third click to represent the partial quantity.

One of the issues with this gesture is if the user tries to make the third click outside the first circle he will end up with a ratio whose value is greater than 1.

Performance Complexity:

The gesture takes constant time O(1) to be performed.

Variations:

None

26. Accumulative Selection and Ordering

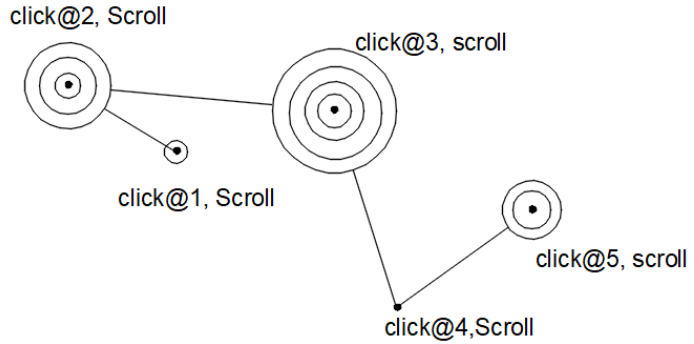


Figure:Gesture to define Ordering while selecting items

Steps to perform:

Click on a point to select and scroll to give an order to it which is default "1" for the first item, and then click next point and scroll again to give the order for this relative to the first one. i.e if we give the second item order "1", then the first item will default to order 2. Similarly if we select the third item and give it order "n", then all the items selected before this with order "n" or "n +p" will move one order up and the items with order "n-1" or less will remain same.

Procedural steps:

OnceOrMore(Once(MouseClick), NoneOrMore(MouseScroll))

Pseudocode:

```
order(from(point@1...n).location.select.data[item],
event(scroll))
```

Application:

This gesture can be used to select items and give an order to them at the same time. While giving the order to the latest selected item, change the order of previous items with respect to the current selected item if necessary.

Example:

This gesture is used by archeologist to select a location and make a plan on the order of sequence in which this this location need to be explored to find interesting items.

This gesture can be used by civil surveyor to select buildings and give it an order based on their construction cost and subsequently to the same for the items he selected or visited in sequence

Usability:

This Gesture is only useful when the items to be ordered are not selected in advance.

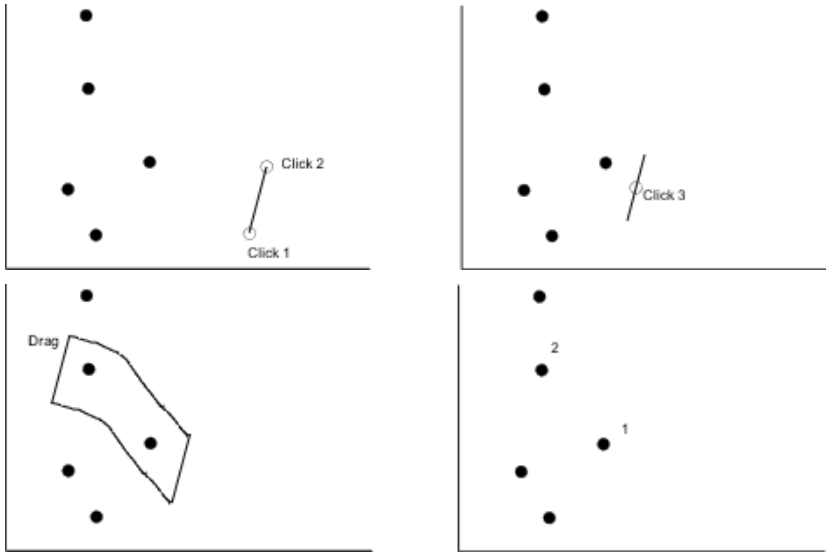
Performance complexity:

The user takes order of n^2 scroll wheel moves and order of n clicks to select n items and order them in worst case scenario.

Variations:

None

27. Ribbon Ordering



Steps to perform:

Two clicks to define the ribbon width and direction. Click to select a starting point, drag to cover the data points, release the click to finish.

Procedural Steps:

Once(MouseClick), Once(MouseClick), Once(MousePressed), OnceOrMore(MouseDragged), Once(MousePressed)

Pseudo Code:

```
line(point@1,point@2).swipe(event.drag()).selectIns
ide.data[item].order()
```

Application:

This gesture is used for giving ordering to several data on a plot by dragging the mouse to form a ribbon. The order starts from the starting point of the ribbon to the ending point.

Example:

The gesture can be used for ordering traveling path of a marketer.

Usability:

The potential problem is when the ribbon swipes multiple data at the same time. In this case, the system might just give priority from top-left side to bottom-right side.

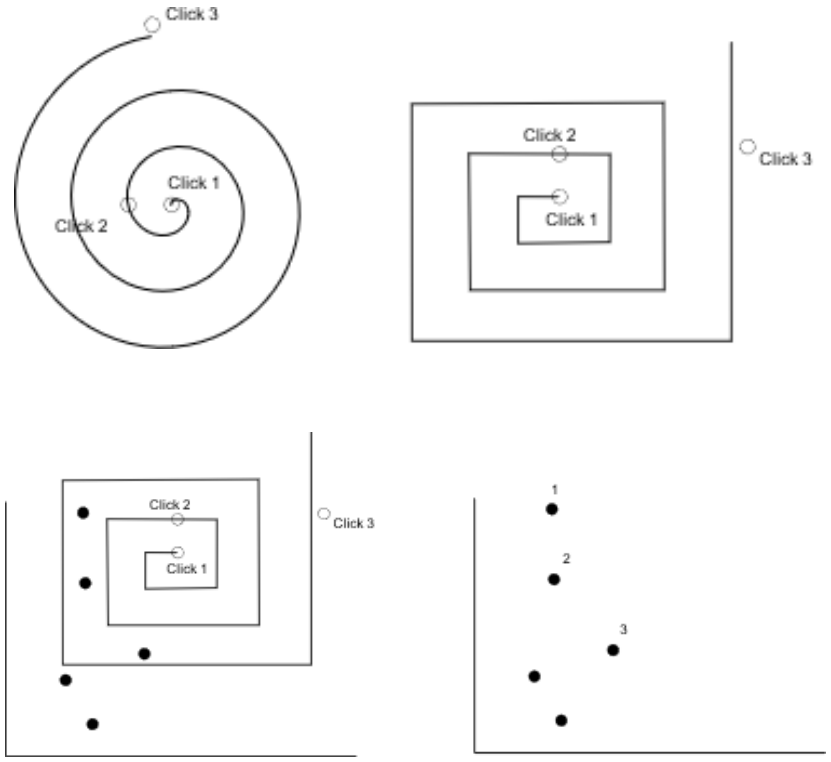
Performance complexity:

The user needs to make two clicks for the ribbon setup. Then, the user needs to drag the ribbon around to cover all data they want to select. Therefore, it has linear complexity (O(N)).

Variations:

None

28. Spiral/Spriangle Ordering



Steps to perform:

Click to select a starting point, second click to define the gap width, third click to define the end point

Procedural steps:

Once(MouseClick), Once(MouseClick), Once(MouseClick)

Pseudo Code:

```
order(spiral(point@1,point@2,point@3).selectInside.
data[item])
```

Application:

This gesture is used for giving ordering data to several data on a plot by. The order starts from the center, extending out to the end point.

Example:

The gesture can be used for ordering geographic based data points. It can effectively divide a region for search effort. The search will start at the center, sprawling out to the outer/end point. An example: when there is a missing person case. The gesture can be used for the search effort by setting the center of the spiral/spriangle to the last known location of the person. Then, the search group can start searching by tracking the location ordered by the gesture.

Usability:

The potential problem is when the gap is too wide, where closer-to-center data gets ordered higher than farther-to-center data. In this case, the user needs to redo the gesture with smaller gap width.

The second click defines the rotation of the spiral/spriangle. To create the axis-perpendicular-spriangle in the figure above, the user must provide the second click strictly vertical from the first click.

The path width is defined by the first and second click.

The third click defines how many loops the spiral/spriangle will have. The gesture will create as many loops until the path reaches just before the third click location.

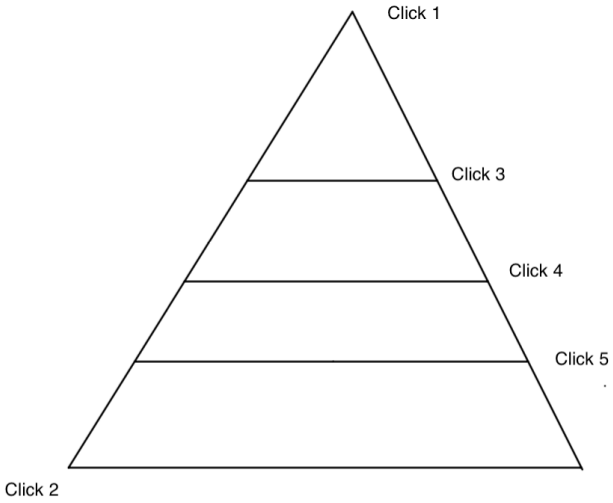
Performance complexity:

Since the user needs to make three clicks for the gesture, then, it has constant complexity.

Variations:

None

29. Triangle for Ordering



Steps to perform:

The first click defines the starting point of a side. The second click defines the end point of the same side. The next set of clicks defines the number of ordering.

Procedural Steps:

Once(MouseClick), Once(MouseClick), NoneOrMore(MouseClick)

Pseudo Code:

triangle(point@1,point@2).orders(from(point@3...n))

Application:

This gesture can be used to order geographic based data points starting from the first click location.

Example:

In the case of javelin throwing sport event, this gesture can be used to rank the throwing result.

Usability:

Based on the number of clicks (from third to the last click) the user can interactively determine the ordering.

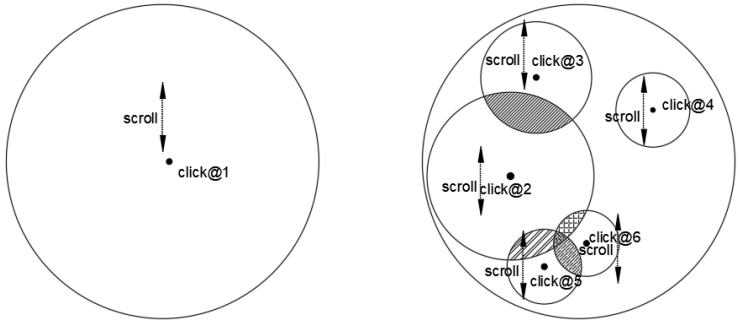
Performance Complexity:

This gesture requires as many clicks as the ordering number to be performed. Therefore, it has O(N) complexity

Variations:

None

30. Venn Diagram Input



Steps to perform:

1. Click to select a starting point, scroll for specifying the size of the outer circle and also the total value.
2. Click inside the outer circle, scroll for specifying the size of the inner circle and also the sub-group value. Type in the label; Enter to finish.
3. Repeat second step as many as needed to create the sub-groups
4. If needed, the user can click and drag the center of an inner circle to move it around

Procedural steps:

Once(MouseClick), OnceOrMore(MouseScroll), OnceOrMore(Once(MouseClick), OnceOrMore(MouseScroll))

Pseudo Code:

```
events.map.circle(p@1,events.type(scroll).amount).p
ut(events.type(mouse).makeCircle(events.clicks,
events.scrolls), events.type(keyboard).
makeLabel(events.type)
```

Application:

This gesture is used for defining dataset that has overlapped values.

Example:

The gesture can be used for inputting data for students who have different hobbies, club activities, etc.

Usability:

The potential problem is when the accuracy of the input is paramount. One way to overcome this problem is by giving visual feedback of the inner circle value whenever the user makes the scrolling event.

Special case(s):

Performance complexity:

This gesture requires as many scroll for defining the values. Therefore, it has O(N) complexity

31. Circle Category Annotation

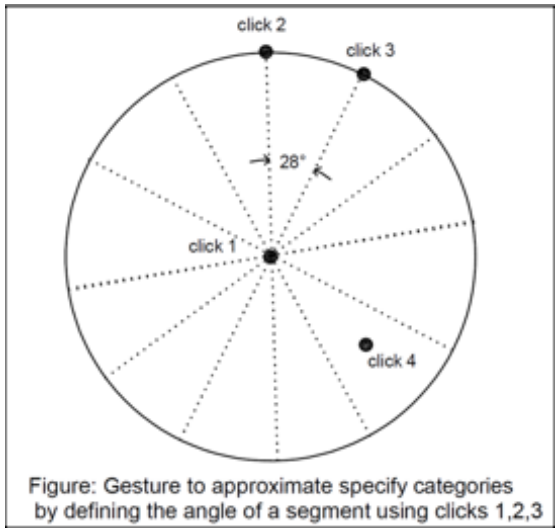


Figure: Gesture to approximate specify categories by defining the angle of a segment using clicks 1,2,3

Steps to perform:

Click any points (click1) in the space to define center of the circle. Click a point (click2) on the plane to define the radius of the circle as well as the reference point. Click a third point on the circle to form an angle with the center and the reference point.

Procedural code:

Once(MouseClick), Once(MouseClick), Once(MouseClick)

Pseudocode:

reference_circle = glyph.circle.radius(click1, click2)

reference_angle = angle(line(click1,click2), line(click1, click3))

circle_sectors = reference_circle.segment.angle((360/ reference_angle).roundoff(roundoff digits))

category_select = reference_circle.circle_sectors.select.start(click2)

Application:

The gesture defined in the above diagrams is used to define approximate number of categories by defining the angle of a sector and using the properties of the circle and later select a category. Once the user enters the angle of the segment, then dividing the total angle of the circle by the angle of the segment specified gives the value which can be rounded off to the closest integer to get the number of categories

Example:

This gesture can be used to identify the categorical data like the variety of crops in a location, the categories of archeological specimens found in a site.

Usability:

The user must make sure to specify angle that would be easy to distinguish cognitively.

For a bigger circle the user can identify a greater number of categories whereas for a smaller circle the user will only identify few categories as the segments of the circle will be hard to differentiate.

Performance Complexity:

Since the gesture only need a fixed number of operations to be performed, it has O(1) complexity.

Variations:

None