GeometryEditor Progress Report for Summer 2007

Xun Lai

The GeometryEditor authoring environment was almost ready for testing and feedbacks from users in early July when I gave my presentation in the summer seminar. As the first testing version of Geometry is near completion, there has to be a plan or roadmap about what new features or improvements should be added immediately, and what should be postponed to the next version.

I have finished the following features and improvements after my last presentation. These are features and improvements that I think should be included in the first testing version.

1. Making the macro algorithm smarter

A macro saves all the steps to create a construction. In a construction, some objects are independent, and some objects depend on other objects. As long as the independent objects are given, a macro can create all the dependent objects automatically. An author doesn’t have to go through all the steps to create those dependent objects.

For example, to construct a circle through three points ABC, you have to create segments AB, and AC, midpoints of AB and AC, perpendicular line of a segment through its midpoint, intersection point of two perpendicular lines, and finally a circle centered at the intersection point, and through one point of A, B, or C.

By using a macro, you simply select points, A, B, and C, and then all the other objects will be created automatically.

My original macro algorithm was able to generate macros based on constructions. However, it’s not smart enough.

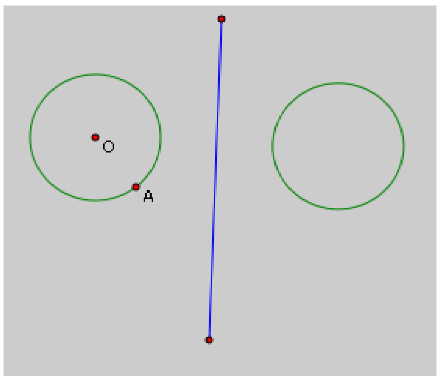


Figure 1

For example, in Figure 1, a segment as a mirror, a circle OA, and its reflected circle about the mirror can define a macro. The reflected circle depends on the mirror line and the source circle.

By using this macro, you can click on a line, and a circle, the reflected circle will be created. My old macro algorithm requires you to always use a circle as a source object for reflection.

However, it's not smart enough. Objects able to be reflected include not only circles, but also points, lines, polygons, and so on. A smarter macro should allow you to select the mirror line, and any object able to be reflected. A reflected object will be generated.

Furthermore, consider another example in Figure 2.

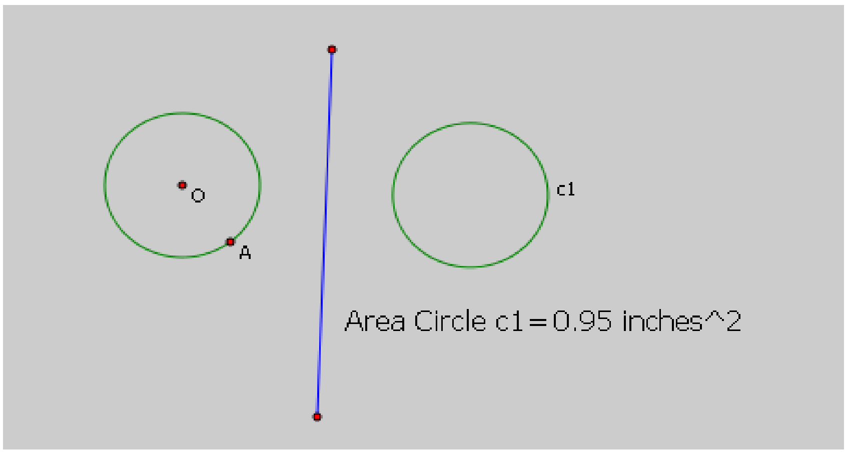


Figure 2

A mirror, a source circle, a reflected circle, and its measured area can define a macro. As in last example, it's not smart either to restrict the reflection source object to be a circle. However, this time, a point or a line could not be the source object because we want to measure the area of the reflected object. Only when the source object is a circle or a polygon, can the reflected object be measured in area.

So my current implementation of the macro tries to find out the broadest possible range of the types of every independent object in a macro. And the types of dependent objects subject to the types of the independent objects.

2. Assigning types to objects

It turns out that simply knowing an object's obvious geometrical types, such as segment, circle, point, polygon and so on is not enough. An object can be of many special types.

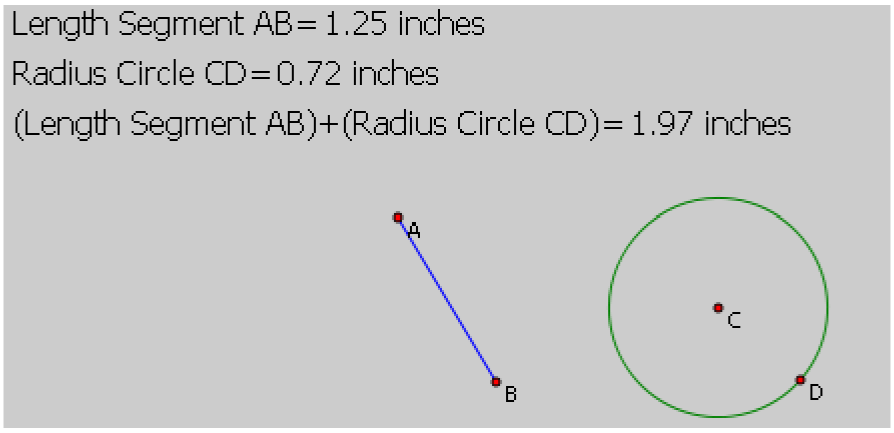
For example, from section 1, a circle can be reflected. So does a line, a point, or a polygon. We can define a type "*Transformable*" to describe these objects. A calculation having a result in distance units such as cm, inches, or pixels can be treated as a "*Distance*" type. A segment can be of both *Transformable* type and of *Distance* type. The type "Areable" can include objects that can be measured in area such as a circle, or a polygon.

Each object has a field to record what types it belongs to. In this field, one bit represents one type. Multiple bits can be 1 at the same time because an object can be of several types.

By assigning types to objects in this way, it solves several problems:

- Making the macro algorithm smarter as in section 1
- Going through objects of a particular type quickly
- Enabling the calculator to use objects as operands directly in an expression

3. Automatic labeling mechanism



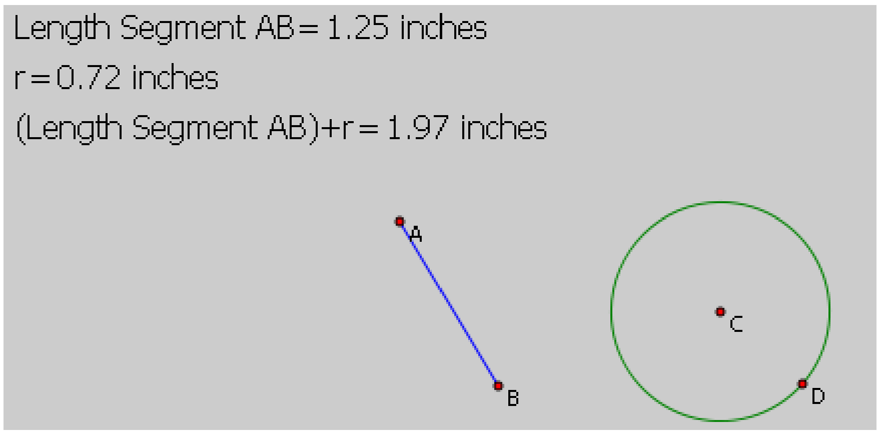
Length Segment AB = 1.25 inches
Radius Circle CD = 0.72 inches
(Length Segment AB)+(Radius Circle CD) = 1.97 inches

Figure 3

When a geometric object is measured, if it doesn't have a user-defined label, the system will assign a label to it automatically. For example, in Figure 3, when the segment was measured, labels A and B were assigned to the two end points of the segment.

When a calculation is defined, the text on the left side of the calculation will generated automatically. In Figure 3, the sum of the length of the segment and the radius of the circle will be given a meaningful label as *(Length Segment AB)+(Radius Circle CD).*

When a label of an object is changed by a user, the system will update all the affected objects.



Length Segment AB = 1.25 inches
r = 0.72 inches
(Length Segment AB)+r = 1.97 inches

Figure 4

For example, if a user changes the measurement label "*Radius Circle CD*" to *r*, the text on the left side of the calculation will be updated to *(Length Segment AB)+r.*

If a label of an object is ever changed, the object's property dialog will allow the author to set the label back to default.

4. Continuing improvements on the calculator.

The calculator is used to define new calculation, new function, and properties of an object. I am continuing to improve its functionalities.

- User-defined function can be used in an new expression
- Segments/circles can be treated as numeric objects to be inserted into an expression.

5. Rewriting and improvement on defining a coordinate system

The old implementation was quite awkward. As shown in Figure 5, when an axis of a coordinate system was defined, a point as the origin, a point controlling the spacing of the markers, a point indicating the end of the positive half of the axis, and a point indicating the end of the negative half of the axis would be created.

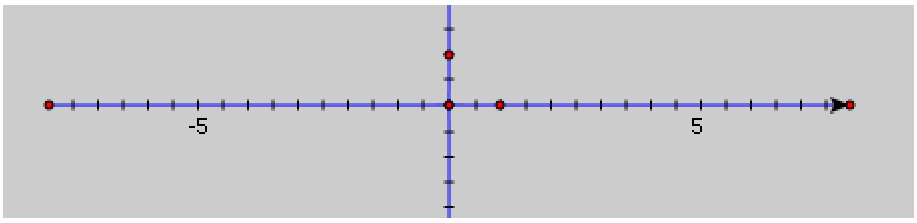


Figure 5

The four points were treated as real points. For instance, an author can use the end point of the positive half of the axis as the center of a circle. Except the origin, it doesn't make much sense to treat the other three points as real points.

Furthermore, if an author is interested to inspect the data file, he or she will be confused that how come lots of objects are defined while I only want to create a coordinate system.

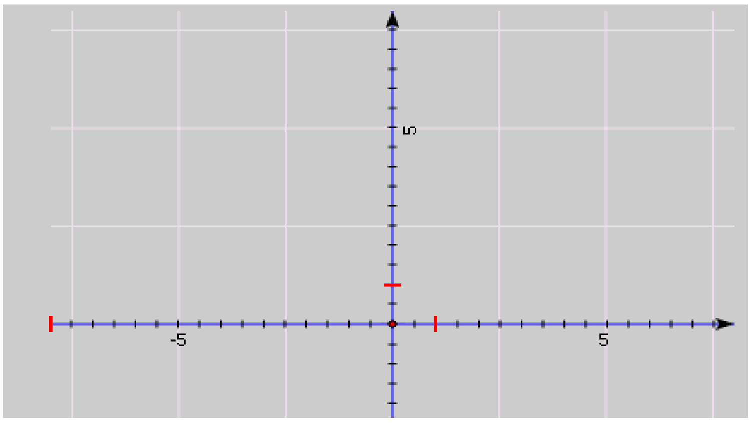


Figure 6

In the new implementation, the points controlling the appearance of an axis are represented as some bars and the arrow. They no longer are saved as separate objects in the data file. It makes the logic of the program clearer, and the data file more understandable by users.

The bar controlling the spacing of the markers provides more powerful control than the previous implementation. Now the spacing can be determined by an expression, can be the same as another axis, or can be dragged freely by a user.

The value between the origin and the marker spacing control bar can be customized too. Sometimes we want extremely small or large values between markers. The old implementation only uses 1 as the value between the origin and the marker control point.

Also grids based on a particular coordinate system are implemented too.

6. User-defined ruler

Previously, only rulers in cm or inches can be created. Now I have finished the codes to allow an author to define a ruler by specifying the spacing between markers, and the value between two markers. The algorithm is similar to the one used in the coordinate system.

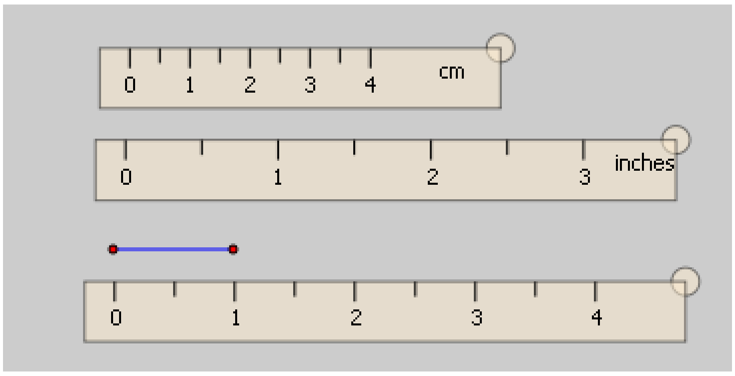


Figure 7

In the last ruler in Figure 7, the length between 0 and the first major marker is determined by a segment.

Also the length of a ruler can be dragged now if an author wants a short ruler or a long ruler.

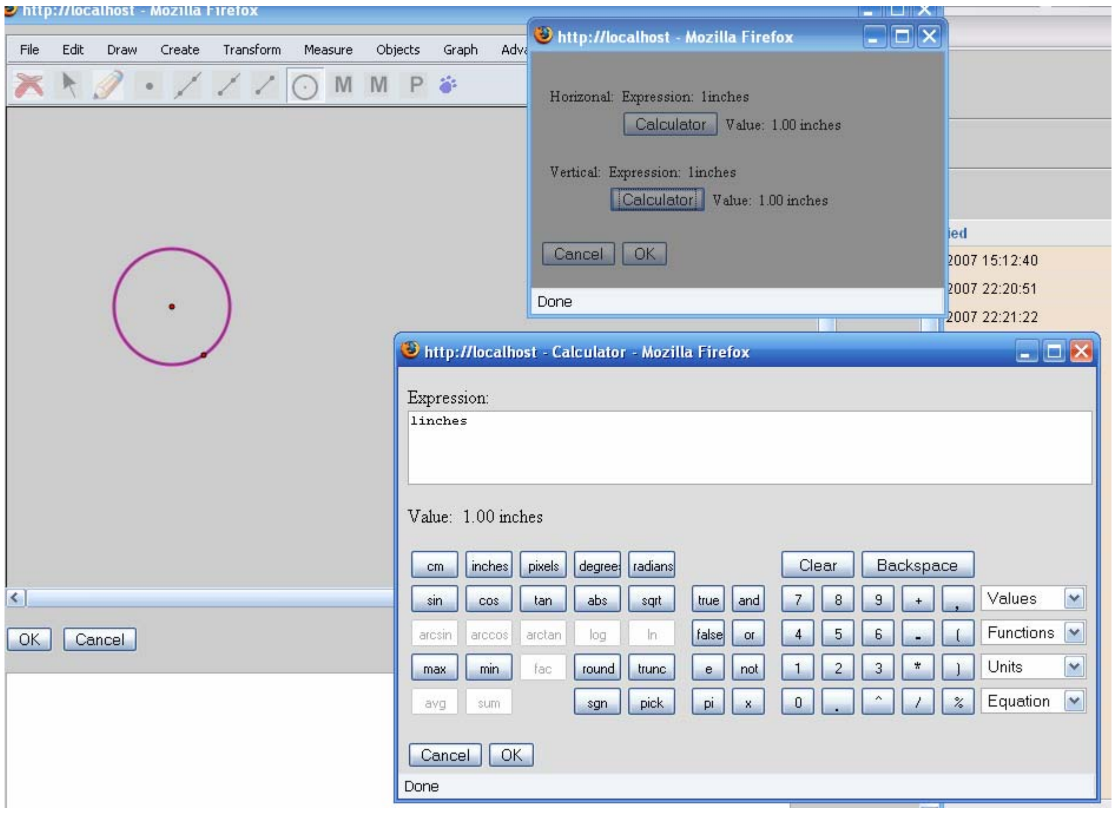
7. Window focus mechanism:

As a Web application, controlling the relation between a pop-up window and its opener is not as easy as one first thought. Also some browsers such as Firefox don’t have modal window support.

The complexity can be:

- All the operations on the opener window should be disabled if it opens a pop-up window.

- Clicking on an opener window should not hide the pop-up window behind the opener.
- Clicking the “Cancel” button and clicking the “X” button on the top right corner of a window should have the same effect.
- Cascading closing of windows. For instance, if window A opens window B, which in turn opens window C. When window A is closed by a user, window B, and C should be closed automatically.



Horizonal: Expression: 1inches
Calculator Value: 1.00 inches

Vertical: Expression: 1inches
Calculator Value: 1.00 inches

Expression:
1inches

Value: 1.00 inches

2007 15:12:40
2007 22:20:51
2007 22:21:22

Figure 8

As in the example shown in Figure 8, the authoring window opens a dialog to define the translation of circle. And the dialog in turn opens the calculator to define a translation distance expression.

Clicking on the authoring window or the first dialog should not hide the calculator behind.

The menus and toolbar in the authoring window were disabled. Operations on the first dialog were disabled. A user is only allowed to interact with the calculator.

If a user closes the first dialog, the calculator should be closed automatically.

Closing the calculator will re-enable the operations in the first dialog. Closing the first dialog will re-enable the menu and toolbar in the authoring window.

I have finished all the algorithms to achieve the above goals. And the algorithms work in general cases instead of any particular dialogs.

9. Adding a lot of dialogs

As a real application, dialogs are important for an author to customize and setting many things.

Here lists the dialogs finished in this summer

- Ruler creation dialog
- Coordinate System creation dialog
- Color setting dialog
- Object style setting dialog
- Global preferences dialog
- Circle with center and radius creation dialog
- Movement definition dialog to customize the speed of an animation
- Several property dialogs for different types of objects.
- Iteration definition dialog

Here lists dialogs implemented before the summer:

- Menu customization dialog
- Toolbar customization dialog
- Translation dialog
- Polar translation dialog
- Vector translation dialog
- Dilation dialog
- Reflection dialog
- Rotation dialog
- Synchronized copy dialog
- Calculator
- Several property dialogs for different types of objects.

10. Other features and improvements

I have also finished several other improvements. You may track my progress by this file

http://wme.cs.kent.edu/geosvg/doc/shorttermTasks.txt

although it's not a well-organized file.

Future Work

The GeometryEditor is ready for testing and feedback from users. I plan to spend one more month to finish basic supports from the Geosite so that people can try the first test version of the GeometryEditor.