The One and Only (Except Source Comments) Guide to OMG!OCRCAL

Abstract

OMG!OCRCAL is a calculator with following near-to-unique features:

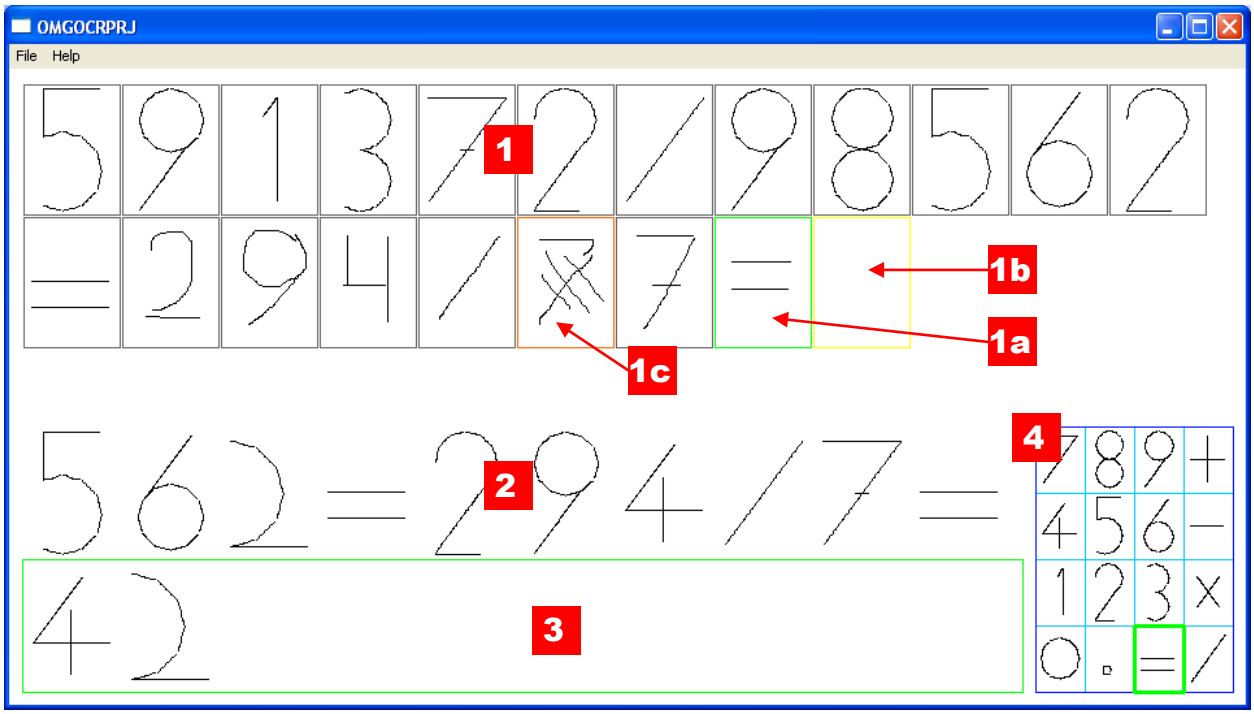
- main input mode is handwriting; keypad provided for accessibility. Section ‘Recognition’ describes certain limitations of handwriting input
- customizable number shapes used both for input and output
- calculation performed on character strings using good old school methods

Running

The program may be started from the Visual Studio IDE or by running the compiled executable file. I recommend running the release version – it responds more smoothly. You should only ensure that the file ‘definition.txt’ is located in the current directory.

Interface

Most of the – may I call it work? – on this program went into creation of user interface.
The main program window is divided into 4 parts:



1. Input area. You may draw shapes by **left-dragging in the green** (1a) **or yellow** (1b) **squares**. Shapes drawn in the green square are added to current image and immediately recognized (see Recognition). When you draw something in the yellow square, the contents of the green one are fixed and yellow square becomes green.
   Characters that don’t fit on screen are skipped, but will appear if you resize the screen.
   Previously recognized characters are marked with gray, and **unrecognized – with red border**. Gray and red squares may not be modified.
   If you find an error in the green square, you may draw multiple lines in the square so that

the shape will be ignored (1c).
The gray and red characters are immutable, bad luck if you haven't noticed the error at once. You may, however, draw '=' to complete the expression and then reenter the characters correctly.

2. Buffer display. Last successfully recognized characters from the input zone are displayed here. **Unrecognized images are silently ignored**.
3. Result display. If the input ends in '=', the result of expression is displayed here. Otherwise, the incomplete expression is shown here. The characters are shrunk if needed to fit the result on screen.
4. Keypad. It is secondary means of input, and much inferior to the handwriting from most points of view, except speed and reliability. Please note that **keypad does not store characters**, but only images. When you press a key on the keypad, the corresponding image is drawn and recognized.
   Besides the character input, the keypad may be used for following actions:
   a. Right-click on '+' makes all characters larger
   b. Right-click on '-' makes all characters smaller
   c. Right-click on '=' resets character size
   d. Right-click on '0' resets all data
   e. Right-click on '4' exits program

Recognition

Basics

Due to the time constraints, fully functional image recognition was impossible. Currently, only the following shapes are recognized:

1. Line
2. Circle
3. Circular segment

Also note that **shapes need to be drawn separately and continuously**. For example, the circles must be drawn in one motion, and '9' and '6' characters may not be drawn in single motion, but instead as a circle first, and then a line. Or vice versa, as you wish.
I'm especially sorry about the inability to enter '8' in a single swift sweep of hand, you'll have to draw one circle above another. Or, as you'll see below, not quite above. And even not necessarily a circle…

Shapes

The algorithm of shape recognition may be quickly described as follows:

1. The shape is a line if distance between any point of shape and a line connecting its start and end is small enough. Also lines are divided into vertical, horizontal, slash- and backslash-like based on the position of their starts and ends.
2. If the shape is not a line, it is probably a circle or a segment thereof. And its center is probably (and I know that it's plain wrong for the segments, but it works due to extremely large tolerance) in the center of the minimal box that may contain the shape. Then we count all angles that are covered by vectors that start in the 'center' and go through the points on shape. The shape is considered a circle if these angles cover all 360 degrees or a segment if there exists a single angle interval. If there are several angle intervals, the shape is something we don't know what to do with. Probably, you tried to draw an '8' as you always do. Sorry once again.

It is clear from the description above that a shape needn't be a circle or a segment in order to be recognized as such. For example, any square will pass a circle test easily, and a '[' shape will be equal to the '(' in the eye of the algorithm.

After we've determined the shape type, we find the 'anchors' – points on the shape that are used to join shapes with each other, forming characters. Top of the line, middle of circle, etc.

Characters

The shape of characters used in OMG!OCRCAL is defined in the file '**definition.txt**'. This file **must be located in the current directory** when you start the program. Its' format is quite human-readable, I encourage you to see it yourself. Special bonus if you'll determine the format of comment string without looking at the code. You may even try and add new shapes of your favorite numbers!
Now, if you've looked at the definition file, you should see that the characters consist of shapes and joins. Some characters have alternate configurations. The characters have distinct names and are addressed by them throughout the program code.

The image recognition process consists of following stages:

1. We collect the list of shapes in image.
2. **If more then one character corresponds to the list**, we check the joins to see which character matches better.

This may give a huge number of false positives, but that's something I decided to live with. Call it optimization.

Character output is controlled by the same rules:

1. A random configuration is selected for the characters with multiple looks.
2. Shapes are drawn.
3. Shapes are pulled together using join information, giving an image
4. Image is fit inside its box

Calculation

The calculation is performed entirely upon character strings without converting them to numbers. The algorithms used are from the elementary school as I remember it. The heart of calculation are **inc()** and **dec()** functions, which return next and previous characters respectively. Other functions use only special characters: '0', '1' and occasionally '9'.
The calculation is performed immediately upon recognition of '=' character in the input area.

Conclusion

Well, that's all. See comments in the code for more information.