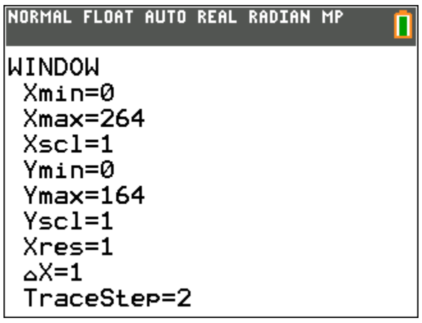
1. Graphing Features

1.1. Basics and Coordinates

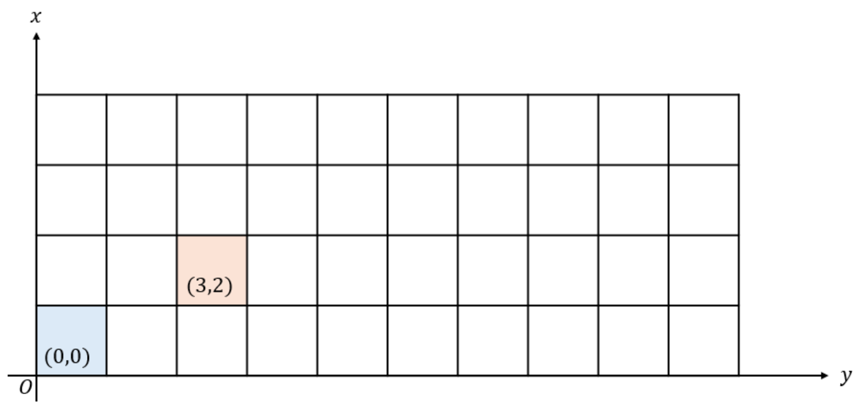


```
NORMAL FLOAT AUTO REAL RADIAN MP
WINDOW
 Xmin=0
 Xmax=264
 Xscl=1
 Ymin=0
 Ymax=164
 Yscl=1
 Xres=1
 ΔX=1
 TraceStep=2
```

Graphing features are exactly the same for the Ti-84 Plus CE and Ti-84 Plus CE Python.

The basic premises of the Ti-84 are first tested. The window is set to the following to ensure maximum resolution, i.e., 165*265.

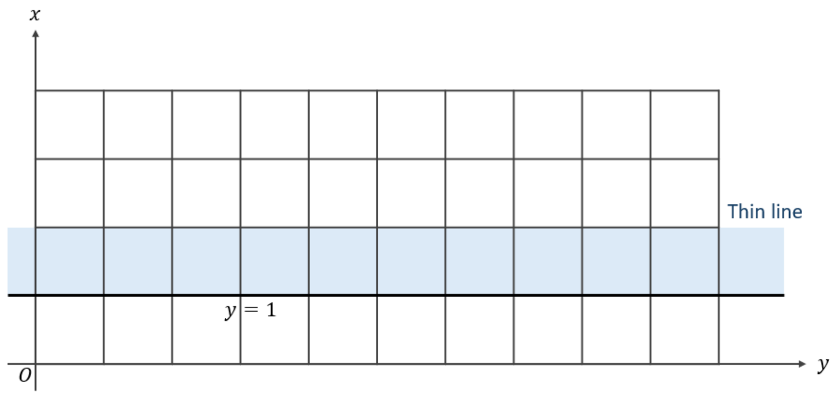
Every pixel is assumed to take its lower left corner as its coordinate.



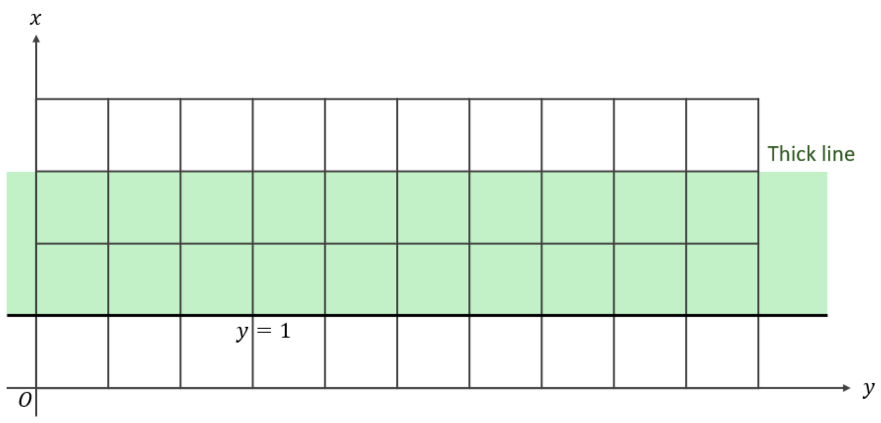
1.2. Lines for graphing equations

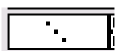
There are 6 types of lines: thin line, thick line, thick dotted line, dotted line, and shade, where the 2 scope types -0 0 are excluded as they don't yield different final results.

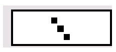
Thin line is a 1-pixel-wide line, thus forming the basis for all other lines.

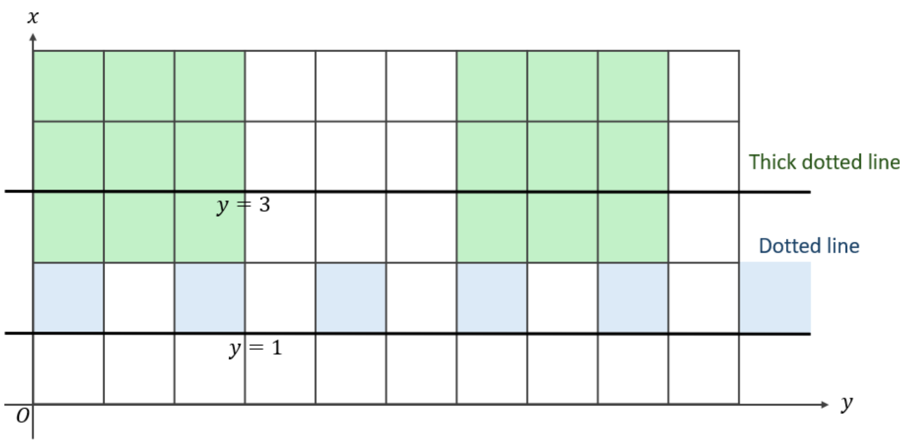


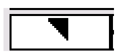
Thick line is a 2-pixel-wide line, which is equivalent to 2 thin lines, with the second thin line placed under the first thin line by 1 pixel.



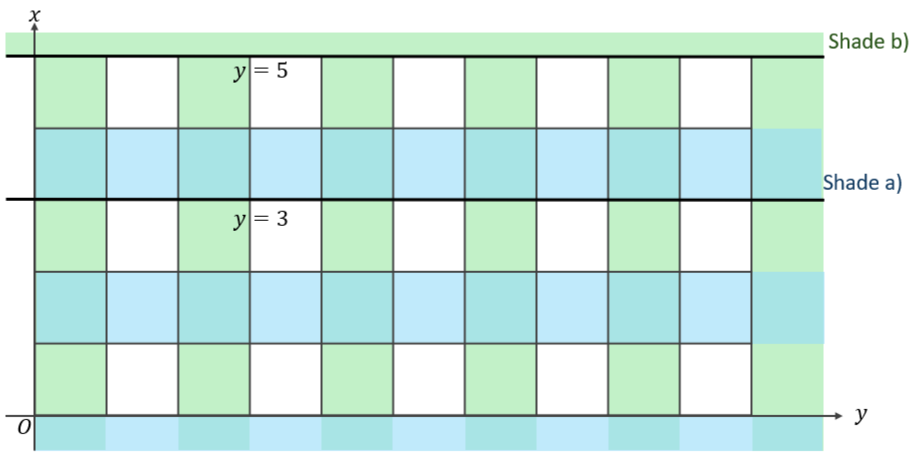
Dotted line  is a thin line but with every other pixel skipped.

Thick dotted line  is a dotted line but with each dot enlarged to a 3 by 3 dot.



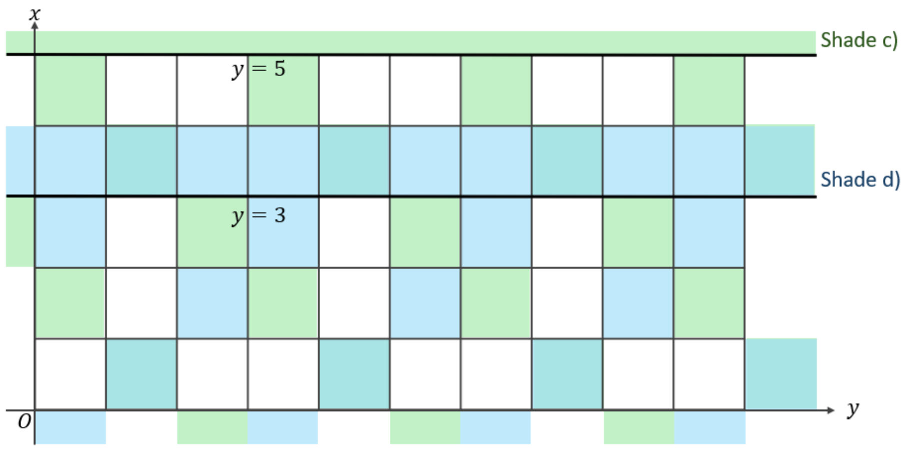
Shade is a thin  line plus a shaded region starting from the next upper/lower pixel. There are 4 types of shade, appearing in order of $x=k, y=k, y=-kx, y=$ kx.

a) $x=k$ leaves the first $x=0$ blank and leaves 1 pixel in between.
b) $y=k$ leaves the first $y=f(x)-1$ blank and leaves 1 pixel in between.



c) $y = -kx$ starts from $(0, f(0) - 1)$ and leaves 2 pixels in between.

d) $y = kx$ starts from $(0, f(0) - 1)$ and leaves 2 pixels in between.



1.3. Plots

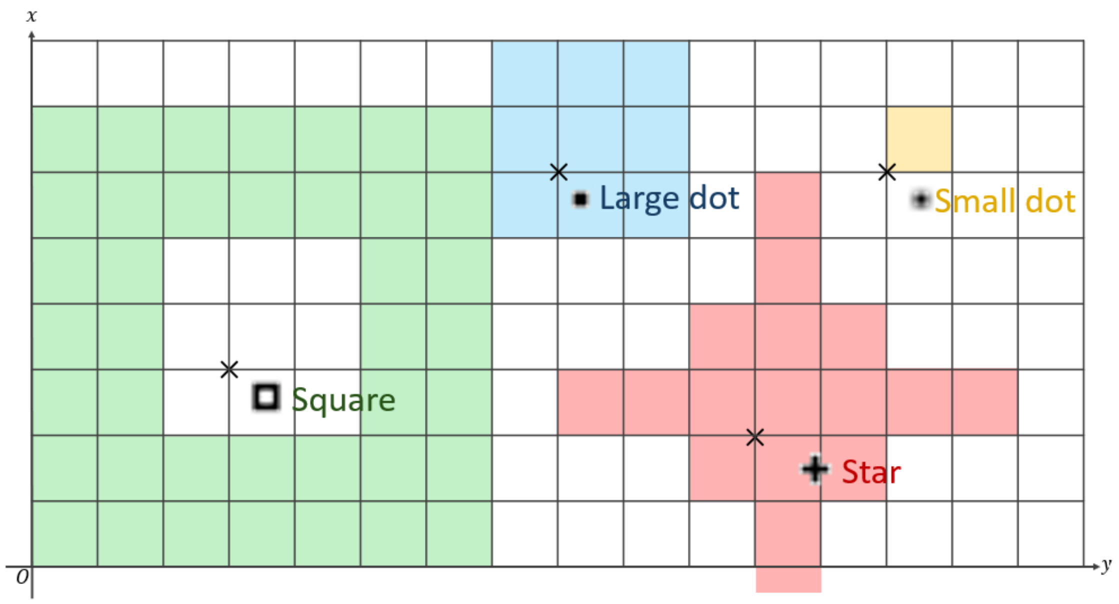
Plot line gives a thick line, and there are 4 types of dots: square, star, large dot, small dot.

Square colours are ± 3 and leave ± 1 blank.

Star colours are ± 1 and the pointers to ± 3.

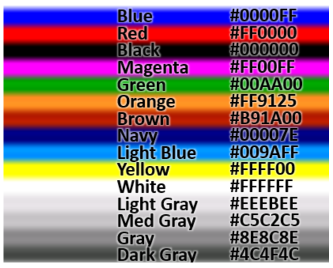
Large dot colours are ±1, while small dots colour the original pixel.

Small dot is a single pixel.



1.4. Colouring

Every colour and their Hex Value are as following:



Blue	#0000FF
Red	#FF0000
Black	#000000
Magenta	#FF00FF
Green	#00AA00
Orange	#FF9125
Brown	#B91A00
Navy	#00007E
Light Blue	#009AFF
Yellow	#FFFF00
White	#FFFFFF
Light Gray	#EEEBEE
Med Gray	#C5C2C5
Gray	#8E8C8E
Dark Gray	#4C4F4C

2. Process

2.1. Introduction

One can categorise parts into three main types based on their size: large, middle, and small.

For large parts, one can begin by formulating their equations, which one can then visualise by plotting them on Desmos. Once plotted, one can transfer them directly to the Ti84.

Middle-sized parts require a more intricate process. Firstly, one can extract key characteristic points of the contours that accurately depict their behaviour. These points are then inputted into Python, where one can use them to derive polynomials. Subsequently, one can use these polynomials to obtain the precise coordinates of the part's contour. Finally, ONE CAN plot these polynomials in Python and then transfer the graph to the Ti84.

For much smaller parts, i.e. parts that are smaller than 10 * 10 pixels, one can opt for a more direct approach. These parts are either hand-drawn or generated using Python directly, depending on the specific requirements.

2.2. Desmos Settings

One can shaded $x < 0, x > 166, y < 0, y > 266$ black to achieve the same screen size as the Ti-84.

2.3. Template Equations

One can first test the equations on Desmos to ensure they are functioning correctly. Once verified, transfer them to my Ti-84 calculator. The following are some commonly used equations with variables to streamline the process of generating new equations for graphs.

$$y = l_{\text{horizontal}}$$
$$x = l_{\text{vertical}}$$
$$y = z\sqrt{(x-p)(q-x)} + c + kx + m\sqrt{(x-m_{\text{left}})(m_{\text{right}}-x)}$$
$$y = \zeta \sin\left(\frac{x}{\rho} + \phi\right) + \omega + \psi x + \frac{(x-\tau)^n}{\xi}$$
$$y = \lambda \sin^{-1}\left(\frac{x-\alpha}{\beta}\right) + \gamma + \delta_1 x + \frac{\delta_2}{x-\alpha-\tau}$$

2.4. Colouring

Since the Ti-84's shading feature doesn't support full colouring, one can use the plot lines feature to achieve a better effect. First use the GC to achieve a close approximation of the desired colour. Then, to replicate this shading on the Ti-84, include dots after shading the main area, effectively mimicking the colour gradient. This process allows me to achieve a visually similar result despite the limitations of the Ti-84's shading capabilities.

i) Employ a Python script to convert Desmos functions into Excel functions. Additionally, adjust equations inward by 1 unit to compensate for the 3-pixel width of plot lines, thereby preventing excessive colouring.

```
import re

def desmos_to_excel(expression):
    expression = expression.replace('x', 'A1')
    expression = expression.replace('sin', 'SIN')
    expression = expression.replace('cos', 'COS')
    expression = expression.replace('tan', 'TAN')
    expression = expression.replace('asin', 'ASIN')
    expression = expression.replace('acos', 'ACOS')
    expression = expression.replace('atan', 'ATAN')
    expression = expression.replace('ln', 'LN')
    expression = expression.replace('log', 'LOG')
    expression = expression.replace('sqrt', 'SQRT')
    expression = expression.replace('abs', 'ABS')
    expression = expression.replace('floor', 'FLOOR')
    expression = expression.replace('ceil', 'CEILING')

    expression = re.sub(r'([a-zA-Z]+)\(', r'\1(', expression)

    return expression

desmos_function = input()
excel_function = desmos_to_excel(desmos_function)
print(excel_function)
```

ii) Use the below programme to convert coordinates from x-coordinate-based to continuous y-coordinate-based representation and align them as left and right boundaries.

```
import sys

xlist, ylist = [], []
with open('temp.txt', 'r') as f:
    for line in f:
        x, y = map(int, line.strip().split())
        xlist.append(x)
        ylist.append(y)

# find max and min y values
y_max = max(ylist)
y_min = min(ylist)

# determine the order of traversal
if ylist.index(y_max) > ylist.index(y_min):
    range_values = range(y_max, y_min - 1, -1)
else:
    range_values = range(y_min, y_max + 1, 1)
```

```
# generate final x and y lists
x_final = [xlist[ylist.index(i)] for i in range_values if i
in ylist]
y_final = list(range_values)

print(y_max, y_min, '\n')
for item in x_final:
    print(item)
```

iii) Transform them into points suitable for plotting lines with the following script. And import to Ti-84s for graph plotting.

```
import csv

input_file = "temp.txt"
output_file = "coordinates.csv"

# read data from the input file
def read_input_data(file):
    x1, x2, y = [], [], []
    with open(file, 'r') as f:
        for line in f:
            x1_temp, x2_temp, y_temp = map(int,
line.strip().split())
            x1.append(x1_temp)
            x2.append(x2_temp)
            y.append(y_temp)
    return x1, x2, y

# generate final coordinates
def final_coordinates(x1, x2, y):
    x_final, y_final = [], []
    for xi1, xi2, yi in zip(x1, x2, y):
        x_final.extend([xi1, xi2])
        y_final.extend([yi, yi])
    return x_final, y_final

#  write coordinates to a CSV file
def write_coordinates(file, x_final, y_final):
    with open(file, 'w', newline='') as csvfile:
        writer = csv.writer(csvfile)
        for x_val, y_val in zip(x_final, y_final):
            writer.writerow([x_val, y_val])

# Main code
x1, x2, y = read_input_data(input_file)
x_final, y_final = final_coordinates(x1, x2, y)
write_coordinates(output_file, x_final, y_final)
```

iv) Archive additional colouring by plotting small dots at desired special density, generated via a shading programme. These coloured dots can then be imported into graphing calculators to modify the graph's appearance.

```
v) import csv

   input_file = 'coordinates.csv'
   output_file = 'shade.csv'
   density = 2

   # read coordinates from file
   def read_coordinates(file):
       x_coords, y_coords = [], []
       with open(file, 'r') as f:
           reader = csv.reader(f)
           for x, y in reader:
               x_coords.append(int(x))
               y_coords.append(int(y))
       return x_coords, y_coords

   # interpolate coordinates based on colour density
   def interpolate_coordinates(x_coords, y_coords,
   density):
       interpolated_x, interpolated_y = [], []
       for i in range(len(x_coords) - 1):
           x1, y1 = x_coords[i], y_coords[i]
           x2, y2 = x_coords[i + 1], y_coords[i + 1]
           delta_x = x2 - x1
           delta_y = y2 - y1
           distance = max(abs(delta_x), abs(delta_y))
           if distance > 0:
               step_x = delta_x / distance
               step_y = delta_y / distance
               for j in range(int(distance)):
                   interpolated_x.append(int(x1 + j *
   step_x))
                   interpolated_y.append(int(y1 + j *
   step_y))
       return interpolated_x, interpolated_y

   # write coordinates to a CSV file
   def write_coordinates(file, x_coords, y_coords):
       with open(file, 'w', newline='') as csvfile:
           writer = csv.writer(csvfile)
           for x, y in zip(x_coords, y_coords):
               writer.writerow([x, y])

   # Main code
   x_coords, y_coords = read_coordinates(input_file)
   x_final, y_final = interpolate_coordinates(x_coords,
   y_coords, density)
   write_coordinates(output_file, x_final, y_final)
```

2.5. Contour and Coordinate Detections

As mentioned earlier, I opt for a more direct approach for much smaller parts, the following are the Python program that I used to generate the coordinates.

```
import cv2
import csv

img = cv2.imread('text.png', cv2.IMREAD_GRAYSCALE)
threshold_lower_bound = 150
```

```python
# detecting coordinates
height, width = img.shape
threshold_coordinates = []
for y in range(height):
    for x in range(width):
        if img[y, x] > threshold_lower_bound:
            threshold_coordinates.append((x, 165 - y))

# write to csv
with open('threshold_coordinates.csv', 'w', newline='') as csvfile:
    writer = csv.writer(csvfile)
    writer.writerows(threshold_coordinates)
```