МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКО”

Факультет прикладної математики

Кафедра системного програмування і спеціальних комп’ютерних систем

Лабораторна робота №1

З дисципліни «Комп’ютерна графіка»
«Алгоритми растрування»

Виконав:
студент ІІІ-го курсу
групи КВ-41
Горпинич-Радуженко Іван

Київ 2016

Текст програми:

```
from __future__ import division
from PIL import Image
from time import *

exp=[]

def _fpart(x):
    return x - int(x)

def _rfpart(x):
    return 1 - _fpart(x)

def putpixel(img, xy, color, alpha=1):

    c = tuple(map(lambda bg, fg: int(round(alpha * fg + (1-alpha) * bg)),
                  img.getpixel(xy), color))
    img.putpixel(xy, c)

def wuline(x1,y1,x2,y2, color,image):
    starclock=clock()
    dx, dy = x2-x1, y2-y1
    steep = abs(dx) < abs(dy)
    p = lambda px, py: ((px,py), (py,px))[steep]

    if steep:
        x1, y1, x2, y2, dx, dy = y1, x1, y2, x2, dy, dx
    if x2 < x1:
        x1, x2, y1, y2 = x2, x1, y2, y1

    grad = dy/dx
    intery = y1 + _rfpart(x1) * grad
    def draw_endpoint(pt):
        x, y = pt
        xend = round(x)
        yend = y + grad * (xend - x)
        xgap = _rfpart(x + 0.5)
        px, py = int(xend), int(yend)
        putpixel(image, (px, py), color, _rfpart(yend) * xgap)
        putpixel(image, (px, py+1), color, _fpart(yend) * xgap)
        return px

    xstart = draw_endpoint(p(*(x1,y1))) + 1
    xend = draw_endpoint(p(*(x2,y2)))

    for x in range(xstart, xend):
        y = int(intery)
        putpixel(image, p(x, y), color, _rfpart(intery))
```

```
            putpixel(image, p(x, y+1), color, _fpart(intery))
            intery += grad
        exp.append({'name':"Wu Algorithm:",'clock':(clock()-starclock)})

    def bresenham_circle(x0, y0, radius, color, image):
        starclock=clock()
        f = 1 - radius
        ddf_x = 1
        ddf_y = -2 * radius
        x = 0
        y = radius
        image.putpixel((x0, y0 + radius), color)
        image.putpixel((x0, y0 - radius), color)
        image.putpixel((x0 + radius, y0), color)
        image.putpixel((x0 - radius, y0), color)

        while x < y:
            if f >= 0:
                y -= 1
                ddf_y += 2
                f += ddf_y
            x += 1
            ddf_x += 2
            f += ddf_x
            image.putpixel((x0 + x, y0 + y), color)
            image.putpixel((x0 - x, y0 + y), color)
            image.putpixel((x0 + x, y0 - y), color)
            image.putpixel((x0 - x, y0 - y), color)
            image.putpixel((x0 + y, y0 + x), color)
            image.putpixel((x0 - y, y0 + x), color)
            image.putpixel((x0 + y, y0 - x), color)
            image.putpixel((x0 - y, y0 - x), color)
        exp.append({'name':"Bresenham Circle",'clock':(clock()-starclock)})

    def test_time():
        ttest=[]
        c=4
        for i in range(1500):
            dda_line(10, 10, 450, 150, black, img4)
            bresenham_line(10,10,450,150,black,img4)
            wuline(10, 10, 450, 150,black,img4)
            bresenham_circle(250,250,100,black,img4)
            bresenham_ellipse(250,250,200,50,black,img4)
        for elem in exp:
            if c>=0:
                ttest.append(elem)
                c-=1
            else:
                for telem in ttest:
```

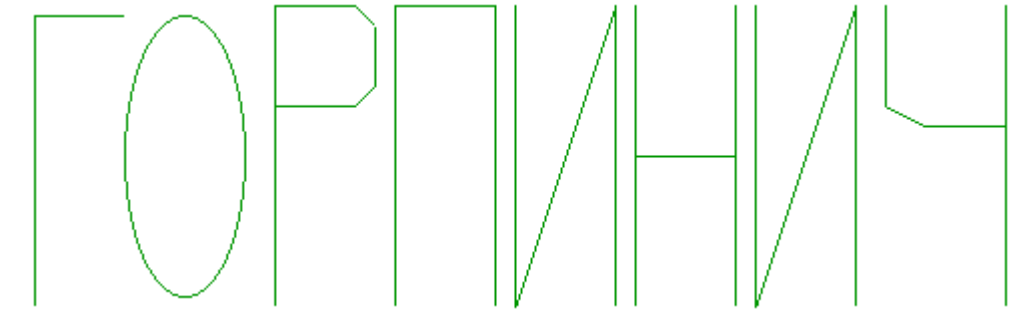
```
            if elem['name']==telem['name']:
                telem['clock']+=elem['clock']
    return ttest

def dda_line(x1, y1, x2, y2, color, image):
    starclock=clock()
    dy = y2 - y1
    dx = x2 - x1
    if abs(dx) > abs(dy):
        steps = dx
    else:
        steps = dy
    xIncrement = float(dx) / float(steps)
    yIncrement = float(dy) / float(steps)
    image.putpixel((x1, y1), color)
    for i in range(steps):
        x1 += xIncrement
        y1 += yIncrement
        image.putpixel((int(round(x1)), int(round(y1))), color)
    exp.append({'name':"DDA Algorithm:",'clock':(clock()-starclock)})

def bresenham_line(x1,y1,x2,y2,color,image):
    starclock=clock()
    dx=abs(x1-x2)
    dy=abs(y1-y2)
    sx=-1
    sy=-1
    if x2>x1:
        sx=1
    if y2>y1:
        sy=1
    error=dx-dy
    while x1!=x2 or y1!=y2:
        image.putpixel((x1,y1),color)
        error_=error
        if error_>-dy:
            error-=dy
            x1+=sx
        if error_<dx:
            error+=dx
            y1+=sy
    exp.append({'name':"Bresenham Algorithm:",'clock':(clock()-starclock)})

def bresenham_ellipse(x0,y0,width,height,color,image):
    starclock=clock()
    a2=width*width
    b2=height*height
    fa2=4*a2
    fb2=4*b2
```

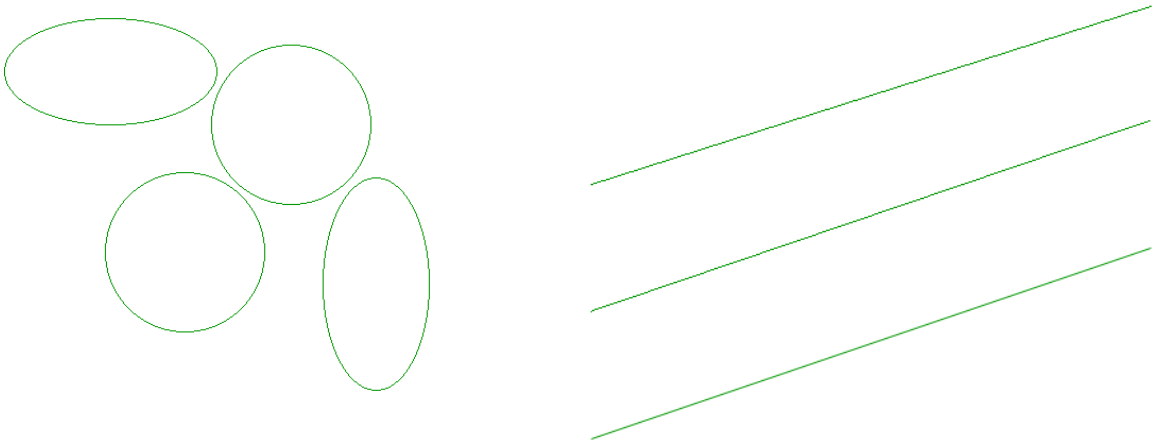
```
    x = 0
    y = height
    sigma = 2*b2+a2*(1-2*height)
    while b2*x <= a2*y:
        image.putpixel((x0 + x, y0 + y), color)
        image.putpixel((x0 - x, y0 + y), color)
        image.putpixel((x0 + x, y0 - y), color)
        image.putpixel((x0 - x, y0 - y), color)
        if sigma>=0:
            sigma+=fa2*(1-y)
            y-=1
        sigma+=b2*(4*x+6)
        x+=1
    x = width
    y = 0
    sigma = 2*a2+b2*(1-2*width)
    while a2*y<=b2*x:
        image.putpixel((x0 + x, y0 + y), color)
        image.putpixel((x0 - x, y0 + y), color)
        image.putpixel((x0 + x, y0 - y), color)
        image.putpixel((x0 - x, y0 - y), color)
        if sigma>=0:
            sigma+=fb2*(1-x)
            x-=1
        sigma+=a2*(4*y+6)
        y+=1
    exp.append({'name':"Bresenham Ellipse",'clock':(clock()-starclock)})

def print_time(expr):
    print "Drawing 1500 lines(in seconds):"
    for alg in expr:
        print alg['name'],
        print alg['clock']

def draw_lastname(color,image):
    bresenham_line(15, 15, 15, 160, color, image)
    bresenham_line(15, 15, 60, 15, color, image)

    bresenham_ellipse(90,85,30,70,color,image)

    bresenham_line(135,10,135,160,color,image)
    bresenham_line(135,10,175,10,color,image)
    bresenham_line(135,60,175,60,color,image)
    bresenham_line(175,10,185,20,color,image)
    bresenham_line(175,60,185,50,color,image)
    bresenham_line(185,50,185,20,color,image)

    bresenham_line(195,10,195,160,color,image)
    bresenham_line(245,10,245,160,color,image)
```

```
    bresenham_line(195,10,245,10,color,image)

    bresenham_line(255, 10, 255, 160, color, image)
    bresenham_line(305, 10, 305, 160, color, image)
    bresenham_line(255, 160, 305, 10, color, image)

    bresenham_line(315, 10, 315, 160, color, image)
    bresenham_line(365, 10, 365, 160, color, image)
    bresenham_line(315, 85, 365, 85, color, image)

    bresenham_line(375, 10, 375, 160, color, image)
    bresenham_line(425, 10, 425, 160, color, image)
    bresenham_line(375, 160, 425, 10, color, image)

    bresenham_line(500, 10, 500, 160, color, image)
    bresenham_line(440, 10, 440, 60, color, image)
    bresenham_line(440, 60, 460, 70, color, image)
    bresenham_line(460, 70, 500, 70, color, image)

    image.save("lastname.png")

def draw_lines(color,image):
    dda_line(10,150,450,10,color,image)
    bresenham_line(10,250,450,100,color,image)
    wuline(10, 350, 450, 200,color,image)
    image.save("Lines.png")

def draw_circles(color,image):
    bresenham_circle(170,270,75,color,image)
    bresenham_ellipse(100,100,100,50,color,image)
    bresenham_ellipse(350,300,50,100,color,image)
    bresenham_circle(270, 150, 75, color, image)
    image.save("Circles.png")

white=(255,255,255)
black = (0, 150, 0)
img1 = Image.new("RGB", (500,400), white)
img2 = Image.new("RGB", (500,500), white)
img3 = Image.new("RGB", (510,170), white)
img4 = Image.new("RGB", (500,500), white)
tttime=test_time()
print_time(tttime)
draw_lines(black,img1)
draw_circles(black,img2)
draw_lastname(black,img3)
```

Скріншоти:

```
Drawing 1500 lines(in seconds):
DDA Algorithm: 2.04484339834
Bresenham Algorithm: 1.51032115731
Wu Algorithm: 13.1925648154
Bresenham Circle 1.75608833034
Bresenham Ellipse 2.65318208779
```





Висновки:

Алгоритм DDA-лінії : Застосування обчислень з дійсними числами і лише одноразове використання округлення для остаточного отримання значення растрової координати зумовлюють високу точність і низьку швидкодію алгоритму.

Алгоритм Брезенхейма малює відрізок дуже швидко, але він не виконує згладжування. Також він не може обробити ситуацію, коли кінцеві точки мають не цілочисельні координати.

Хоча **алгоритм Ву** також часто використовується в сучасній комп'ютерній графіці через підтримку згладжування, алгоритм Брезенхейма залишається вживаним завдяки його швидкісті і простоті.