Scan Conversion

Question 1 *Develop a Bresenham–like algorithm for scan converting the parabola* $y = x^2/2a$ *for* $-a \le x \le a$*. You may assume that* a *is an integer. Your algorithm should use only integer variables and addition. Please be sure to provide a diagram to illustrate how your algorithm selects pixels.*

Main Ideas:

- draw only one half of the parabola ad use symmetry to get the other. For example, $0 \le x \le a$.
- The slope of the parabola in the specified region is less than 1 since

$$\frac{dy}{dx} = x/a \Rightarrow 0 \le \frac{dy}{dx} \le 1 \quad \text{for } 0 \le x \le a$$

  Thus on every step x is incremented and the question is if to increment y or not.
- Draw a diagram to decide which pixels you want to set (Figure 1). Forget about integer/floating numbers in the beginning, you should be able to normalize variables later on to enforce this condition.

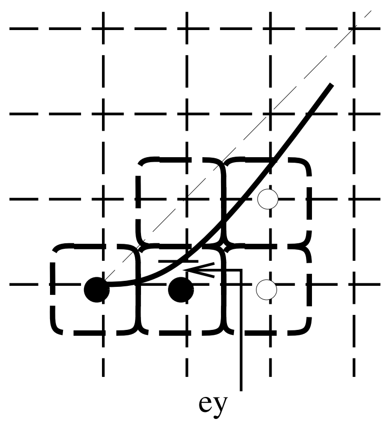

Figure 1 Picking pixels for scan converting the parabola

- Bresenham's algorithm (and the solution) is incremental. You start with some initial x, y and error in y, call it ey, and update them on each iteration.
- The increment is :

$$\frac{(x+1)^2}{2a} - \frac{x^2}{2a} = \frac{2x+1}{2a}$$

The resulting algorithm before massaging it to get all–integer algorithm:

```
x = 0
y = 0
SetPixel(x, y)
ey = 0
for x = 1 to a do
  ey = ey + (2(x-1)+1)/2a
  if (abs(ey) > 1/2)
        y = y + 1
        ey = ey - 1
  SetPixel(x, y)
```

The variable ey is the only non-integer variable in the last algorithm. We replace it everywhere with an integer variable $iey = ey * 2a$ and normalize to get:

```
x = 0
y = 0
SetPixel(x, y)
iey = 0
for x = 1 to a do
  iey = iey + (2x-1)
  if (abs(iey) > a)
        y = y + 1
        iey = iey - 2a
  SetPixel(x, y)
```

All multiplications in the last version are multiplications by 2 which can be achieved by shifts. □

Clipping:

Question 2 *Construct an algorithm to clip lines to a circular window. The circle is defined by a radius R and a center point $C = (a, b)$ lying in the xy–plane. The lines are defined by their end points. Develop all the formulas you need for quick accept reject tests and intersection computations.*

Main Ideas:

- Since eventually we'll need to compute intersections it is a good idea to keep one representation implicit and the other parametric. A reasonable choice is to pick parametric representation for the line and implicit one for the circle. Thus if the two points are (x_1, y_1) and (x_2, y_2) then the equation of the line is:

$$L(t) = (x_1, y_1) + t((x_2 - x_1), (y_2 - y_1)) \quad 0 \leq t \leq 1$$

  The equation of the circle:

$$C(x, y) = (x - a)^2 + (y - b)^2 - R^2 = 0$$

- It is important to notice the difference between a *line* and a *line segment*. What we're clipping are line segments and not lines of infinite length.

Different cases arise as shown in Figure 2,

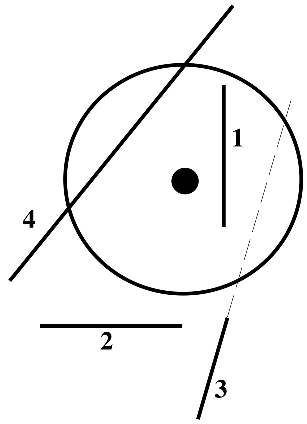

Figure 2 Different cases for clipping to a circle

Quick Accept: If both end points are inside the circle (case 1 in Figure 2) then the whole line segment is accepted and no clipping is needed, that is accept whole line segment if

$$\begin{aligned} &(x_1 - a)^2 + (y_1 - b)^2 \leq R^2 \qquad \text{and} \\ &(x_2 - a)^2 + (y_2 - b)^2 \leq R^2 \end{aligned}$$

Quick Reject: The obvious mistake is to assume that if both end points are outside the circle then the whole line segment is rejected (See case 4 in figure).

Many possible fast reject tests are possible, one reasonable test would be to check that the distance between the line going through the points (x_1, y_1) and (x_2, y_2) and the center of the circle is greater than the radius of the circle R. The following diagram explains distance computations:

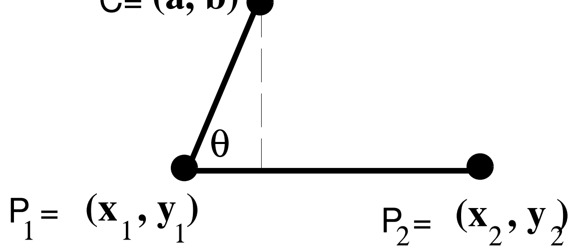

Figure 3 The distance between C and P_1P_2 is $|C - P_1| \sin \theta$.

Thus the condition on the distance (squared because the distance above is signed) is that

$$\frac{||(C - P_1) \times (P_2 - P_1)||^2}{||P_2 - P_1||^2} > R^2.$$

The same formula for the distance can be obtained by computing (double) the area of $\triangle(a, b), (x_1, y_1), (x_2, y_2)$ divided by the length of the vector $(x_1, y_1), (x_2, y_2)$. That is:

$$\frac{\begin{vmatrix} 1 & 1 & 1 \\ x_1 & x_2 & a \\ y_1 & y_2 & b \end{vmatrix}^2}{(x_2 - x_1)^2 + (y_2 - y_1)^2} > R^2.$$

Clipping Line Segment: At this point either the line segment really intersects the circle (case 4 in figure) or the infinite line containing the specified segment does but the segment itself doesn't (case 3 in Figure 2). To get the intersection points we plug the parametric representation of the line into the implicit representation of the circle, and solve for the parameter value t. Substituting, we get:

$$(x_1 + t(x_2 - x_1))^2 + (y_1 + t(y_2 - y_1))^2 - R^2 = 0$$

which reduces to

$$t^2((x_2 - x_1)^2 + (y_2 - y_1)^2) + t(2x_1(x_2 - x_1) + 2y_1(y_2 - y_1)) + (x_1^2 + y_1^2 - R^2) = 0$$

Now solve the quadratic equation in t for the two parameter values corresponding to the intersection points (generally a line intersects a circle in two points), only the segment with $t \in [0, 1]$ lies on the specified line segment and should be kept.$\square$

Fractals:

Question 3 *Starting with the outline of a square, consider iterating the procedure which replaces each straight line segment by a square bump whose sides are 1/3 the size of the original segment (see Figure 4).*

a. *Write a program to draw the N^{th} level of this curve.*

b. *Compute the following:*

  i. *The fractal dimension of the curve.*

  ii. *The perimeter of the curve.*

  iii. *The area inside the curve.*

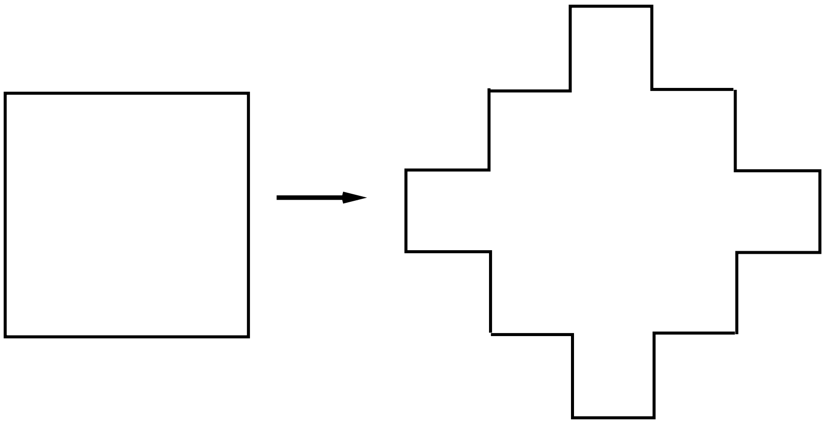

Figure 4 A square bump placed on each side of a square.

Main Ideas: This is a bread and butter Fractals question. If you know the material well it's a straightforward question.

a. A Program: If you write a turtle program, it will be a slight variant of the following program:

```
Frac(N)
  If (N = 1)
        Forward 1
  else
        Frac(N-1)
        Left 90
        Frac(N-1)
        Right 90
        Frac(N-1)
        Right 90
        Frac(N-1)
        Left 90
        Frac(N-1)

Fractal(N)
        Frac(N) Right 90 Frac(N) Right 90 Frac(N) Right 90 Frac(N)
```

For coordinate dependent implementations, you need to compute the five affine transformations that take a line (x_1, y_1) into a line with a bump. Since the lines are only vertical and horizontal this is easy.
A Scheme flavored program (well, minus 1000 ('s,)'s and using infix notation!)

```
G0 = {{(0, 0), (1, 0)}, {(1, 0), (1, 1)},{(1, 1), (0, 1)}, {(0, 1), (0, 0)}}
```

```
Fractal(N)
  if (N = 1) return G0
  else
        Return(Map(Bump, Fractal(N-1)))

Bump(line={(x1, y1), (x2, y2)})
     Return({{(x1, y1),
           (x1 + (x2 - x1)/3, y1 + (y2 - y1)/3)},
        {(x1 + (x2 - x1)/3, y1 + (y2 - y1)/3),
           (x1 +(y2 - y1)/3 + (x2 - x1)/3, y1 + (y2 - y1)/3 + (x2 - x1)/3)},
        {(x1 +(y2 - y1)/3 + (x2 - x1)/3, y1 + (y2 - y1)/3 + (x2 - x1)/3),
            (x1 +(y2 - y1)/3 + 2(x2 - x1)/3, y1 + 2(y2 - y1)/3 + (x2 - x1)/3)},
        {(x1 +(y2 - y1)/3 + 2(x2 - x1)/3, y1 + 2(y2 - y1)/3 + (x2 - x1)/3)
            (x1 + 2(x2 - x1)/3, y1 + 2(y2 - y1)/3)},
        {(x1 + 2(x2 - x1)/3, y1 + 2(y2 - y1)/3), (x2, y2)}})
```

- Fractal Dimensions: By definition,

$$\begin{aligned}
\text{Fractal Dimensions} &= -\frac{\log(n)\text{umber--of--copies}}{\log(S)\text{caling factor}} \\
&= -\frac{\log 5}{\log 1/3} \\
&= \frac{\log 5}{\log 3}
\end{aligned}$$

- Perimeter: If we start with a line segment of unit length and repeatedly introduce bumps into it, the length grows as:

  $$1, 5/3, (5/3)^2, \ldots$$

  Thus, the perimeter is eventually infinite.

- Area: Try the first couple of recursive calls and see how the formula develops. The area grows as (assuming a unit square to begin with):

$$\begin{aligned}
\text{Area} &= 1 + 4[1/9 + 1/9(5/9) + \cdots \\
&= 1 + (4/9)\sum_{i=0}^{\infty}(5/9)^i \\
&= 1 + (4/9)/(1 - 5/9) = 2
\end{aligned}$$

Lighting Models:

Question 4 *a. Explain the difference between ambient light, diffuse reflection, and specular reflection.*

b. Consider Figure 5 below. Give formulas for the contributions to the overall intensity at P of the ambient light, the diffuse reflection, and the specular reflection in terms of the vectors L, V, N and the intensity I of the light source.

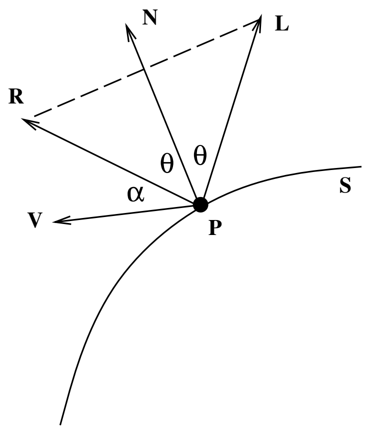

Figure 5 At the point P on the surface S, the vector L points to the light source, the vector V points towards the viewer's eye, and the vector N is normal to the surface S.

Main Ideas:

- Just try to avoid time consuming writing. You're not asked to write down everything that you know about all possible lighting models; just differences, the main differences.
- For the second part, only the formulas for the contributions of different light models are needed.

a. **Ambient Light:** A *constant*–intensity light source that collectively accounts for all kinds of reflected, diffuse light. It is uniform over all the surfaces in the scene.

Diffuse Reflection: A point light source whose rays emanate uniformly in all directions from a single point. The object's brightness varies from one part to another, depending on the direction of and distance to the light source.

Specular Reflection: This can be observed on shiny surfaces. A highlight effect is caused by specular reflection.

b. Contribution of each type of light:

Ambient Light:

$$I = k_u I_u. \quad k_u \text{ is a material dependent ambient reflection coefficient}$$

Diffuse Reflection; The contribution of this component (using the parameters and vectors shown in Figure) is:

$$I = I_p k_d \cos\theta. = I_p k_d (N \cdot L),$$

where I_p is the intensity of the point–light–source and k_d is the diffuse reflection coefficient of the material.

Specular Reflection: The reflectance vector is first computed as:

$$R = 2N(N \cdot L) - L.$$

Now $\cos\alpha = R \cdot V = (2N(N \cdot L) - L) \cdot V$, and the specular reflection contribution is

$$I = k_s I_p \cos^n \alpha, \quad \text{for some } n.$$