The Convergence of IDLA to a Circle

Jean-Luc Thiffeault and Ruojun Wang

August 31, 2018

1 An IDLA Simulation and the Boundary

1.1 An IDLA Simulation with N Particles

Internal diffusion-limited aggregation (IDLA) is a cluster growth process in which particles start at one or more sources within a cluster, diffuse outward, and are added to the cluster at the first site outside it they reach. In this section, I summarize how the codes I was given simulate an IDLA process, and how I modify them to see a slightly different IDLA process.

1.1.1 Particles move in 8 directions

I started from MATLAB codes given by Professor Thiffeault. To beign, a grid quadrant is prepared based on the total number of particles which involve in the process:

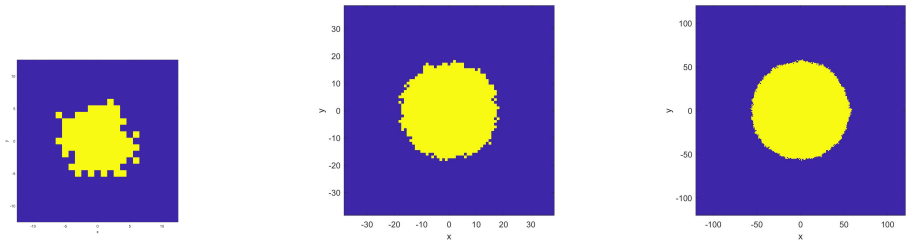
```
Ngrid = ceil(1.2*sqrt(Npart));
    grid0 = Ngrid+1;
    grid = zeros(2*Ngrid+1);
```

where `Npart` represents the total number of particles that take part into the simulation. `Ngrid` gives the length of one side of grid quadrant; `grid0` is the center of the grid quadrant and generate an array with size `2*Ngrid+1` and `grid` generates a `2*Ngrid+1*2*Ngrid+1` matrix consisting of zero . Core codes to simulate IDLA process are as following:

```
drift = [0 0];
for i = 1:Npart
      X = [0 0];
  while 1
        X = X + (randi(3,1,2)-2) + drift;
        if ~grid(X(1)+grid0,X(2)+grid0)
          grid(X(1)+grid0,X(2)+grid0) = 1;
          break
    ...
```

Inside the for loop, we start the first particle from the location `X = [0 0]`, which would locate at the center `grid0` of the quadrant we set previously. Then inside the while loop `X` generates the coordinate that the next particle would locate. As hoe particles in an IDLA process behave, the next particle would move due to a random direction which is generated by `(randi(3,1,2)-2)`. Explicitly, a particle can choose one of eight directions to move along: 2 horizontal directions, 2

vertical directions, and 4 diagonal directions. A `drift` might be added to effect the final shape of the occupied region; the direction of drift can be given by `drift = [0 0]` before the for loop. The codes in if statement guarantee that the particle would keep moving due to `(randi(3,1,2)-2)` until it finds an unoccupied grid to settle down. An occupied grid would be marked as "1" then. The results of the simulation are shown in Figure 1.1. We can observe that with the growth of the number of particles participating in the IDLA process, the boundary of the occupied region tends to imitate a circle.



(a) `Npart = 100` (b) `Npart = 1000` (c) `Npart = 10000`

Figure 1.1: IDLA simulation with 8 directions

1.1.2 Particles move in 4 directions

My first task is to make particles move in 4 directions randomly rather than in 8 directions and to see if the final shape of the occupied region still converges to a circle. The diagonal motion of particles (i.e. upper left, upper right, lower left, lower right) are eliminated in this case. The MATLAB codes to realize this are:

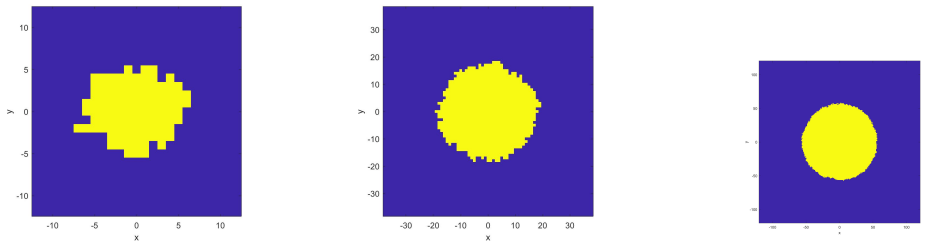
```
v_dir = [1 0; 0 1; -1 0; 0 -1];
n_dir = 4;

for i = 1:Npart
  X = [0 0];
  while 1

  d = randi(n_dir);

  X(1) = X(1) + v_dir(d, 1) + drift(1);
  X(2) = X(2) + v_dir(d, 2) + drift(2);
  ...
```

Here, we prepare a $4 * 2$ matrix containing 4 vectors including four directions that a particle can move along. Then similarly as the process with 8 possible directions in the previous section, inside the for loop, we start from the center of the grid quadrant `[0 0]`. After determining a random number inside a list consisting 1 to 4, we are able to select a vector in the 4*2 matrix and then determine the direction of the particle moves along. The results of the simulation are shown in Figure 1.2.



(a) `Npart = 100` (b) `Npart = 1000` (c) `Npart = 10000`

Figure 1.2: IDLA simulation with 4 directions

We could then observe that an IDLA simulation generates an occupied region which converges to a circle, as the number of particles which participate in the process grows (Figure 1.2).

1.2 Boundary of the Occupied Region

Next, we select out the pixels which constitute the boundary of the occupied region. By studying these pixels, we might understand to what extent the final occupied region converges to a cirlce. Three different algorithms are designed to give the boundary only as the IDLA process evolves.

1.2.1 Three Algorithms to Build the Boundary

Build entire boundary (Figure 1.4(a)). We start by examining an occupied grid p_i on the boundary (for example, one on the boundary which has the same horizontal coordinate as the center grid). Then by checking its neighbor occupied grids, we choose the next grid p_{i+1} which locates on the boundary. By examining all neighbors of p_{i+1} and keep determining the next grid, we find all the grids on the boundary of the occupied region.

```
for k=1:size(bdP,1)
  if ~(bdP(k,1) == 0 && bdP(k,2) == 0)
  finish = 0;
  neighb = [bdP(k,1)+1 bdP(k,2); bdP(k,1)-1 bdP(k,2);
    bdP(k,1) bdP(k,2)+1; bdP(k,1) bdP(k,2)-1;
    bdP(k,1)+1 bdP(k,2)+1; bdP(k,1)+1 bdP(k,2)-1;
    bdP(k,1)-1 bdP(k,2)+1; bdP(k,1)-1 bdP(k,2)-1];

    for j=1:8
      if (grid(neighb(j,1), neighb(j,2))==1 &&
      gridB(neighb(j,1), neighb(j,2))==0)

        if ~((grid(neighb(j,1)+1, neighb(j,2))==1) &&
        (grid(neighb(j,1)-1, neighb(j,2))==1) &&
        (grid(neighb(j,1), neighb(j,2)+1)==1) &&
        (grid(neighb(j,1), neighb(j,2)-1)==1))
```

```
bdP = [bdP; neighb(j,1) neighb(j,2)];
gridB(neighb(j,1), neighb(j,2))=1;
...
```

In the first for loop, we prepare the coordinates of all eight neighbors for one occupied grid on the boundary: Since the boundary of the occupied region can be constructed by grids which are only connected diagonally, so to examine the total eight closest grids for one is necessary. If an occupied grid is marked as "1" and an unoccupied one is marked as "0," we can determine the next occupied grid on the boundary by seeing if all its horizontally and vertically connected neighbors are marked as "1."

Build boundary incrementally (Figure 1.4(b)). This algorithm is designed to check all the grids on the grid quadrant and select out the grids which construct the boundary only as the occupied region grows. The idea is to come up with a method to tell the difference between the grids inside the region and outside it: the grids which are not on the boundary are surrounded by grids along its four sides, while the grids which are on the boundary have at least one side which does not touch another grid. Hence, as one more particle is added to the process, we could eliminate those which are not considered on the boundary. For example, in Figure 1.3, when the fifth particle is added to the process, we can determine the grid at the center is no longer a grid on the boundary and it should be removed. The MATLAB codes are shown as following:

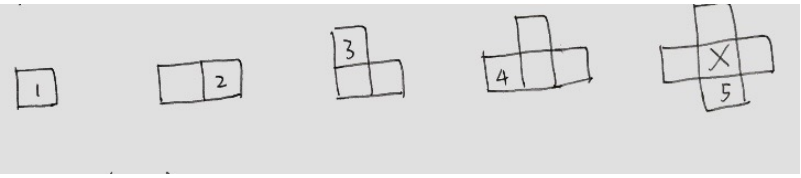


Figure 1.3: Build boundary incrementally: When the fifth particle is added to the process, a grid which is not on the boundary is then appears and needs to be removed.

```
for k = 2:(2*grid0-1)
  for l = 2:(2*grid0-1)
    if grid(k, l) == 1 && grid(k, l+1) == 1 &&
    grid (k, l-1) == 1 && grid (k+1, l) == 1 &&
    grid (k-1, l) == 1

      gridB(k, l) = 0;
      ...
```

where the grids which are occupied are marked as "1," and those which are not occupied are denoted as "0." By examining if all four neighbors of one grid are marked as "1," we are able to determine whether one grid is considered on the boundary.

Build boundary by Discrete Laplace (Figure 1.4(c)). The third algorithm using Discrete Laplace has the similar idea to the second one. A Discrete Laplace can be written as:

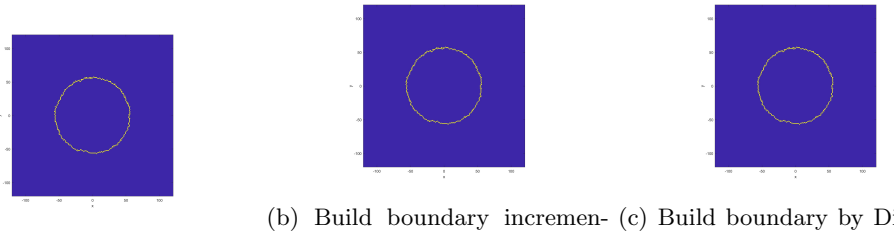
$$(\Delta f)_{i,j} = 4f_{i,j} - f_{i+1,j} - f_{i-1,j} - f_{i,j+1} - f_{i,j-1}, \tag{1.1}$$

where i and j represent the horizontal and vertical coordinates of the center of one grid. Similarly to the second algorithm, the grids which are occupied are marked as "1," and those which are not occupied are denoted as "0." Hence, if $(\Delta f)_{i,j} > 0$, the grid with coordinate (i, j) is considered on the boundary. The MATLAB codes can be written as:

```
for k = 2:(2*Ngrid)
  for l = 2:(2*Ngrid)
    delta_f = 4*grid(k, l) - grid(k, l+1) - grid(k, l-1)
    - grid(k+1, l) - grid(k-1, l);

    if delta_f == 0
      gridB(k, l) = 0;
      ...
```

In order to see if those three algorithms return the same results regarding the grids on the boundary which are selected out, I also further compare the discretized boundary generated by three algorithms and it turns out they agree with each other.



(a) Build entire boundary.

(b) Build boundary incrementally.

(c) Build boundary by Discrete Laplace.

Figure 1.4: Boundary of the occupied region resulted by IDLA process.

1.2.2 The Numerical Standard Deviation

We compute a numerical average distance $\overline{d_N}$ based on the distance between the center of each grid on the boundary and the center of the grid quadrant, as each time one more particle is added to the process, where N denotes the particle number in the process. This distance $\overline{d_N}$ can be roughly considered as the radius of a circle O, if we could obtain such an O when the number of particles participating in the IDLA process $N \to \infty$. From Figure 1.5, we see that $\overline{d_N}$ and N are roughly linearly related. We further compute the standard deviation σ_N, with respect to the particle number N. The relation of σ_N versus N can also be plotted by MATLAB (Figure 1.5). When $N = 10000$, $\sigma_N = 0.72172$. If the occupied region generated by IDLA process converges to a circle as we expected previously, σ_N should be also converging as a smooth function with respect to a large N; however, it is hard to see if σ_N converges from the plot. We need to examine further.

We denote this σ_N as σ_{sim}, saying that this is the standard deviation given by the whole simulation. It consists of two different errors, the geometric error σ_{geom} and the statistical error

σ_{stat}. We would first look at the the geometric error σ_{geom}, since it helps us understand how big this value this even if we draw a perfect circle on a grid quadrant.

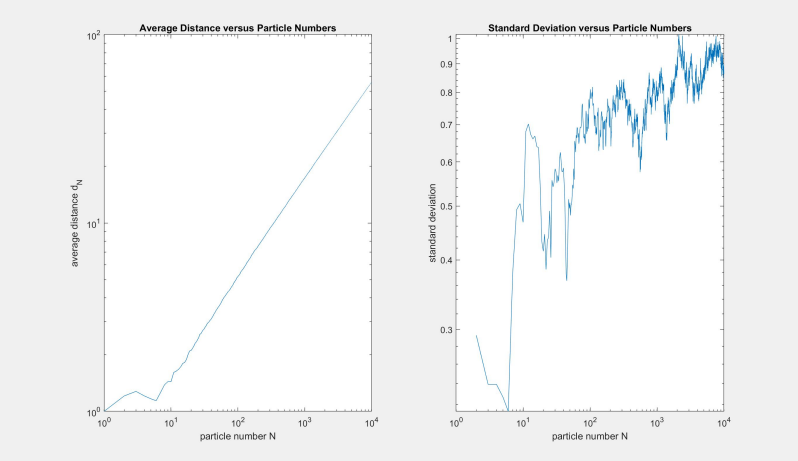


Figure 1.5: Average distance versus particle numbers and standard deviation versus particle numbers when `Npart=10000`.

2 The Geometric Error

The basic setup is given by Professor Thiffeault's handout "On Discretizing a Circle". We consider a grid (a lattice of circle) in the plane with centers at coordinates $p = (m, n) \in \mathbb{Z}^2$. The center of circle drawn locates at the origin $(0, 0)$. Take a circle of radius R, centered on the origin. A continuous discretization of the circle is an ordered set of distinct pixels

$$\mathcal{D}_R = (p_i)_{0 \le i \le N-1} = (m_i, n_i)_{0 \le i \le N-1}, \tag{2.1}$$

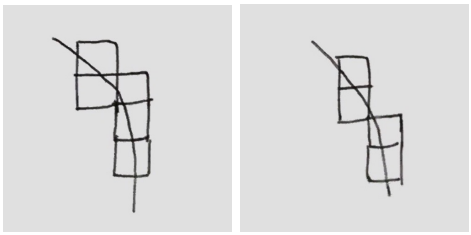
where m_i denotes the horizontal coordinate of the center of a pixel and n_i denotes the vertical one of that. There are two different ways to construct such a discretized circle by coding. For convenience, we call them "non-remove" case and "remove" case.

2.1 Two Types of Discretized Circle and Their Numerical Standard Deviation

2.1.1 The "Non-remove" Case

In a "non-remove" case, the pixels p_i is just those which the boundary of the circle centered at $(0, 0)$ passes through. The resulted boundary consists of the grids connected horizontally, vertically, and diagonally (Figure 2.1).

The algorithm can also be realized by MATLAB codes. The main idea is to start from a grid on the boundary, and to examine its neighbors and to determine the next grid which could be used to



(a) The "non-remove" case.

(b) The "remove" case.

Figure 2.1: Construct the discretized boundary of "non-remove" and "remove" cases

construct the boundary of a circle. We only consider grids for the upper right quarter of the circle; other three discretized quarters would just be the mirror images of the upper right one respect to the horizontal or vertical axises in the center. The final constructed circle is shown in Figure 2.2. We can then compute a numerical standard deviation σ_{geom}. As an example, given $R = 10000$, $\sigma_{geom} = 0.3729$.

2.1.2 The "Remove" Case

Another ways to construct a discretized consisting of less pixels on the boundary than those in the first algorithm: The boundary of the cirlce might or might not pass through each pixel, but no grids on the boundary can have more than one horizontally or vertical neighbors.

The algorithm can be realized by MATLAB codes: The main procedures are similar to those for a "non-remove" circle; when we examine neighbors of one particular grid, we only remain the neighbor whose center is closest to the boundary of the circle which passes through (Figure 2.3). The numerical standard deviation can be computed: When $R = 10000$, $\sigma_{geom} = 0.2624$, which is smaller than the value obtained from the first algorithm.

2.2 The Upper Bound of L_2 Error of the Discretization

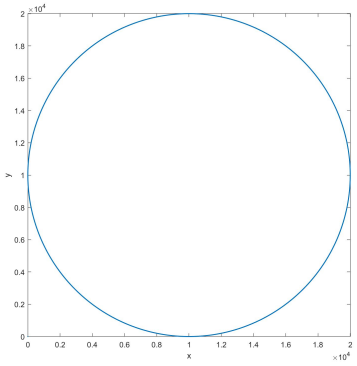
A derivation of L_2 Error of the discretization is provided by Professor Thiffeault's handout "On Discretizing a Circle". Once we have $\mathcal{D}_R = (p_i)_{0 \le i \le N-1} = (m_i, n_i)_{0 \le i \le N-1}$, we define the average radius of $\mathcal{D}_R$ as

$$\text{Rad } \mathcal{D}_R = \frac{1}{N} \sum_{i=0}^{N-1} (m_i^2 + n_i^2)^{\frac{1}{2}}. \tag{2.2}$$

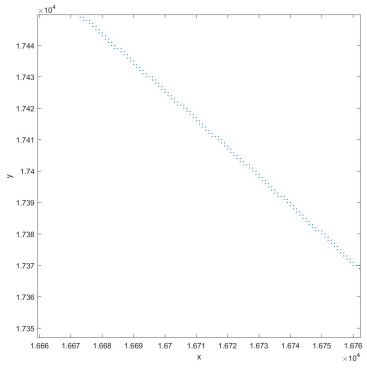
In order for a discretization to be valid, it must converge to the circle in the limit as $R \to \infty$: (?)

$$\lim_{R \to \infty} \text{Rad } \mathcal{D}_R = R. \tag{2.3}$$

The L^2 error can then be written as



(a) A "non-remove" circle.



(b) A "non-remove" circle with $\times 10^4$ zoom in.

Figure 2.2: A "non-remove" circle, its $\times 10^4$ zoom in. Circle gridpoints rather than square colored grids are used to raise the computation speed.

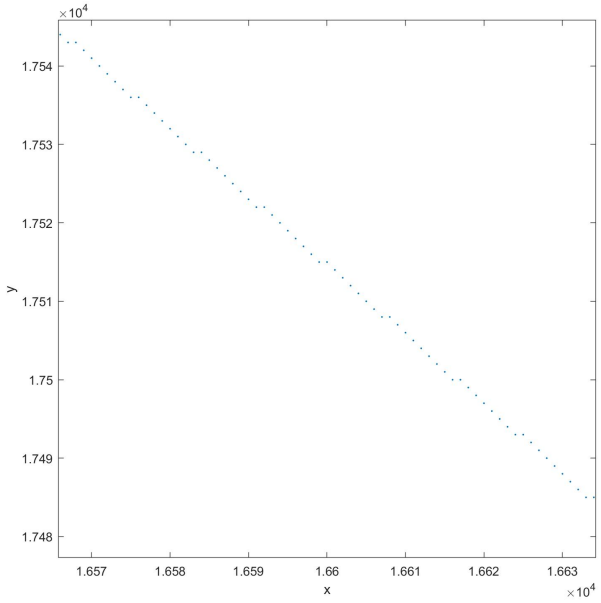


Figure 2.3: A "remove" circle's $\times 10^4$ zoom in.

$$\text{Err}_2\ \mathcal{D}_R = (\frac{1}{N}\sum_{i=0}^{N-1}(m_i^2 + n_i^2) - R^2)^{\frac{1}{2}}. \tag{2.4}$$

In both algorithm to construct a discretized circle, σ_{geom} can be approximated by $\text{Err}_2\ \mathcal{D}_R$ when $R \to \infty$. $\text{Err}_2\ \mathcal{D}_R$ can also be considered as the distance between the center of each pixel p_i and the boundary of the circle which passes through this pixel. We denote $\text{Err}_2\ \mathcal{D}_R$ as d in this case and denote a small piece of the boundary passing through each circle as c_i. As $R \to \infty$, c_i could be just a segment rather than an arc of the circle. Assume that this piece c_i is randomly distributed in each grid; we could then compute the expectation value $\mathbb{E}\ d^2$: Suppose a small region inside a grid with area $dx \times dy$ such that x represents the horizontal coordinate and y represents the vertical one. The probability density function $p(x, y)$ is then given by

$$p(x, y) = \frac{1}{A} = 1 \tag{2.5}$$

where A represents the area of one grid and $A = 1$. $p(x, y)$ should also fulfill $\int_0^1 \int_0^1 p(x, y)\ dx\ dy = 1$. Also, with

$$d^2 = (x - \frac{1}{2})^2 + (x - \frac{1}{2})^2, \tag{2.6}$$

the expectation value of d^2 is then

$$\mathbb{E}(d^2) = \int_0^1 \int_0^1 f(x,y)\ p(x,y)\ dx\ dy = 2\ \mathbb{E}(x - \frac{1}{2})^2 = 2\int_0^1 (x - \frac{1}{2})^2 \cdot 1\ dx = \frac{1}{6}. \tag{2.7}$$

Thus, if the small piece c_i is randomly distributed, the expectation value $\mathbb{E}\ d^2 \approx 0.1667$.

For both "non-remove" and "remove" cases, an upper bound can be found for d^2, which approximately equal to σ^2_{geom} when $R \to \infty$. For now, by observation, $\sigma^2_{geom} \leq \frac{1}{2} \approx 0.5$ and then $\sigma_{geom} \leq 0.7071$, since the largest possible d^2 equals to $\frac{1}{2}$. Our next task is to find a smaller upper bound for σ_{geom} and obtain a new bound which is as close to 0.1667 as possible. We firstly sort pixels into 4 different types as following to lower the upper bound of d^2.

2.2.1 Sort Pixels into 4 Different Types

According to the ways that the boundary of the circle passes through each pixel, we can sort N pixels into 4 different types for the upper right quarter of the circle (other three quarters would just be the mirror images of the upper right one with respect to different axises of symmetry). As the algorithm provided above, the center of each pixel is represented by (m_i, n_i). The boundary of the circle passing by would have 2 different intersections with a pixel p_i (Figure 2.4). Hence, the four different type of grids can be given by the inequalities restricting the coordinates (x, y) of the intersections. For example, the intersection point A in the first kind of pixel is provided by

$$x = m_i - \frac{1}{2};\ n_i - \frac{1}{2} \leq y \leq n_i - \frac{1}{2},\ \text{where } y = \sqrt{R^2 - (m_i - \frac{1}{2})^2}; \tag{2.8}$$

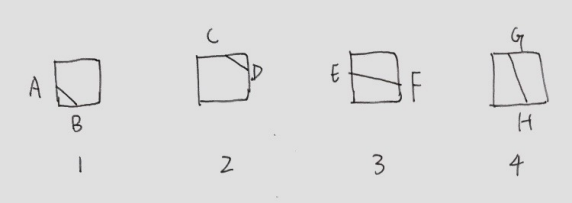


Figure 2.4: Sort pixels into 4 different types

Other similar inequalities can be written to represent other types of pixels. We guess that these four different types of pixels might hold different upper bounds of L^2 error. Hence, by weighing four different values of upper bound correctly with respect to the fractions of these types of pixels, we might be able to obtain a smaller upper bound for σ_{geom}. With MATLAB codes, when $R = 10000$, the numerical fraction of the four different type of pixels in the first quarter for a "non-remove" case are

$$\text{fraction of type 1} = 0.29291; \tag{2.9}$$
$$\text{fraction of type 2} = 0.29286; \tag{2.10}$$
$$\text{fraction of type 3} = 0.20711; \tag{2.11}$$
$$\text{fraction of type 4} = 0.20711; \tag{2.12}$$

For a "remove" case, this would be

$$\text{fraction of type } 1 = 0.20522; \tag{2.13}$$

$$\text{fraction of type } 2 = 0.20897; \tag{2.14}$$

$$\text{fraction of type } 3 = 0.29291; \tag{2.15}$$

$$\text{fraction of type } 4 = 0.29291; \tag{2.16}$$

A derivation for upper bound of the error. $Q_{m,n}$ denotes the distance between the center of the circle and the center of one pixel (corresponding to $\overline{d_N}$ defined in IDLA process); R is the radius of the circle (Figure 2.5). Then

$$|Q_{m,n} - R| \leq \frac{1}{\sqrt{2}}. \tag{2.17}$$

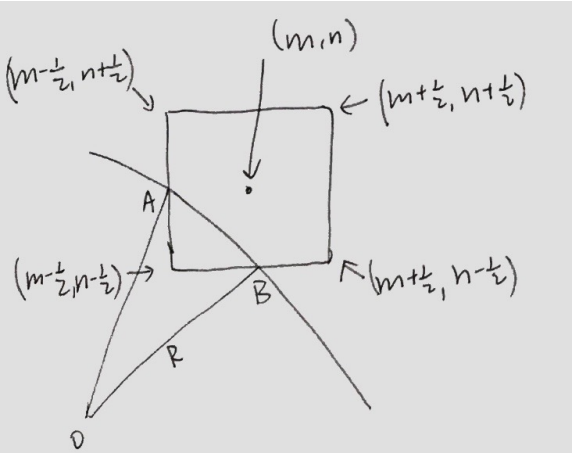


Figure 2.5: Inequalities for each intersection can be written based on the geometry.

The inequalities satisfied by each type of pixels can also be written down in terms of a relation between radius R and the distance $D(x, y)$ between the center of the circle and an intersection between the boundary of the circle and one side of the pixel square. For the intersection point A in a type 1 pixel, we can write

$$D_{m-\frac{1}{2},n-\frac{1}{2}} \leq R \leq D_{m-\frac{1}{2},n-\frac{1}{2}} \tag{2.18}$$

If we just look at $D_{m-\frac{1}{2},n-\frac{1}{2}} \leq R$ and subtract its both sides by $Q_{m,n}$

$$D_{m-\frac{1}{2},n-\frac{1}{2}} - Q_{m,n} \leq R - Q_{m,n}, \tag{2.19}$$

For the left hand side of the inequality, we can compute a general form of difference $D_{m+a,n+b} - Q_{m,n}$ since with it is more convenient to find all four inequalities with this form. Also, $m = Q_{m,n} \cos\theta_{m,n}$ and $n = Q_{m,n} \sin\theta_{m,n}$, substitute those two expressions into (2.12), we obtain

$$\begin{aligned}&\sqrt{(Q_{m,n}\cos\theta_{m,n}+a)^2+(Q_{m,n}\sin\theta_{m,n}+b)^2}-Q\\&=Q(\sqrt{1+\frac{2}{Q}(a\cos\theta+b\sin\theta)+\frac{a^2+b^2}{Q^2}}-1).\end{aligned}\tag{2.20}$$

By Taylor expansion, we have

$$D_{m+a,n+b}-D_{m,n}=a\cos\theta+b\sin\theta+\mathcal{O}(Q^{-1})\leq R-Q_{m,n}.\tag{2.21}$$

Hence,

$$Q_{m,n}-R\leq D_{m,n}-D_{m-\frac{1}{2},n-\frac{1}{2}}\leq\frac{1}{2}(\cos\theta_{m,n}+\sin\theta_{m,n})+\mathcal{O}(R^{-}1).\tag{2.22}$$

$$\Rightarrow\ -\frac{1}{\sqrt{2}R}\leq\frac{Q_{m,n}}{R}-1\leq\frac{1}{\sqrt{2}R}.\tag{2.23}$$

By the same procedures, we can obtain the inequality (2.16) with other three different $D(a,b)$. According to the definition of variance σ^2, we have

$$\sigma^2_{m,n}=\mathbb{E}[Q^2_{m,n}-R^2].\tag{2.24}$$

Square both sides of (2.16),

$$\frac{Q^2_{m,n}}{R^2}-\frac{2\ Q_{m,n}}{R}+1\leq\frac{1}{2R^2}\Rightarrow\frac{\mathbb{E}[Q^2_{m,n}]}{R^2}-\frac{2\ \mathbb{E}[Q_{m,n}]}{R}+1\leq\frac{1}{2R^2}\Rightarrow\frac{\mathbb{E}[Q^2_{m,n}]}{R^2}-1\leq\frac{1}{2R^2}.\tag{2.25}$$

Hence,

$$\sigma^2_{m,n}\leq\frac{1}{2}\approx 0.5.\tag{2.26}$$

Hence, $\sigma_{geom}\leq 0.7071$. The upper bound is still the same as we speculate previously. We need to find another method to get a smaller bound.

2.2.2 An Observation of the Relation between the Fraction of Each Type of Pixels and Multiples of R

This is an observation from examining the numerical values of a particular multiple of radius R and the fraction of each type of pixels. The fraction of each type of pixels is listed in the previous section. We shall look at the geometry of a quarter itself. In Figure 2.6 segment CD is perpendicular to segment OA, where O is the center of the circle. We obtain $OD=\frac{R}{\sqrt{2}}$ and $AD=R(1-\frac{1}{\sqrt{2}})\approx 0.29289R$, where R is the radius of the circle.

To construct the relation, we start by examining how each type of grids functions to help the boundary grows from A to C. For a "non-remove" case, the vertically coordinate increases for each pixel; if R is very large, the total increment is approximately $\frac{R}{\sqrt{2}}$. The horizontal coordinate decreases by $R(1-\frac{1}{\sqrt{2}})$ in total. We will only look at the pixels in the first octant since the second

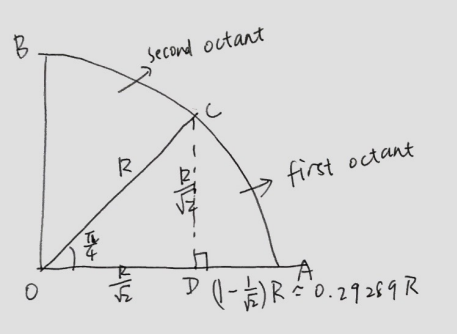


Figure 2.6: The geometry of the upper right quarter and the first octant.

octant is just the mirror image of the first one; hence, we just need to consider type 1, 2, and 4, because type 3 would not appear in the first octant. The fractions ($R = 10000$) for each type in the first octant are given by

$$\text{fraction of type 1} = 0.2929; \tag{2.27}$$

$$\text{fraction of type 2} = 0.2929; \tag{2.28}$$

$$\text{fraction of type 4} = 0.4142; \tag{2.29}$$

Type 1 and 4 result in vertical increments by 1 pixel; type 2 results in horizontal decrement by 1 pixel. Hence, in the first octant,

$$\text{number of type 1 and 4} = \frac{R}{\sqrt{2}} \approx 0.70711R; \tag{2.30}$$

$$\text{number of type 2} = R(1 - \frac{1}{\sqrt{2}}) \approx 0.29289R; \tag{2.31}$$

Hence, when $R \to \infty$, multiples of R help determine the fractions of each type of grids.

For a "remove" case, from Figure 2.6, we observe that the numerical value represents segment AD might relate with the fraction of type 3 and 4 in a "remove" case given by (2.15) and (2.16). However, the same method used to derive a relation does not apply to a "remove" case, since now how each type of grids contributes to a particular increment or decrement is no longer clear. Thus, we need to sort grids in a more specific way.

2.2.3 Sort Pixels into 6 Different Types

Instead of just sorting pixels into 4 types, we sort those in the first octant into 6 types now, in order to examine more on their functions which cause increments and decrements (Figure 2.7). Thus, how each type of pixels causes increment or decrement is determined. If we denote the total horizontal decrement as Δx and the total vertical increment as Δy, then

$$\Delta x \approx n_{1b} + n_{2a} + n_{2b} + n_{4b}; \tag{2.32}$$

$$\Delta y \approx n_{1b} + n_{1a} + n_{2b} + n_{4a} + n_{4b}; \tag{2.33}$$

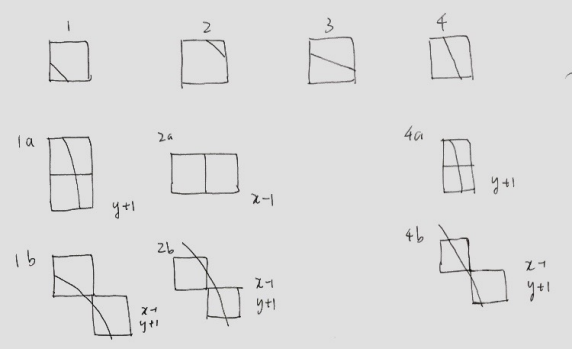


Figure 2.7: Sort pixels into 6 different types.

where n_m gives the number of one type of pixels shown in the first octant. As we show previously, $\Delta x = (1 - \frac{1}{\sqrt{2}})R$ and $\Delta y = \frac{R}{\sqrt{2}}$. Substitute these two into the previous equations and simplify, we get

$$n_{4a} + n_{1a} - n_{2a} \approx (\sqrt{2} - 1)R. \tag{2.34}$$

To examine the number of type 4, we modify () by constructing a $n_{4a} + n_{4b}$ term on the left hand side

$$n_{4a} + n_{4b} \approx (\sqrt{2} - 1)R + (n_{4b} - n_{1a}) + n_{2a}, \tag{2.35}$$

where $n_{2a} = 0$ for the first octant. Also, since every time a 4b type appears, a 1a types appears before it; similarly, when a 1a type shows, a 4b type also shows as its previous pixel. Hence, we have $n_{4b} - n_{1a} = 0$. Then we just leave with $n_4 \approx (\sqrt{2} - 1)R$. The total grids in the first octant n_{total} is just $\frac{2R}{\sqrt{2}}$. The fraction of type 4 in the upper right quarter is

$$\frac{n_4}{n_{total}} = \frac{(\sqrt{2} - 1)R}{\frac{2R}{\sqrt{2}}} = 1 - \frac{1}{\sqrt{2}}. \tag{2.36}$$

Hence, when $R \to \infty$, fraction of type 4 is just $1 - \frac{1}{\sqrt{2}}$. The upper bound of error of type 3 and 4 pixels is $\frac{1}{2}$. Let p_x be the fraction for one kind of pixels; we can weigh a smaller upper bound of d^2 by

$$d^2 \leq p_1 (\frac{2}{\sqrt{2}})^2 + p_2 (\frac{2}{\sqrt{2}})^2 + p_3 (\frac{1}{2})^2 + p_4 (\frac{1}{2})^2 = (\sqrt{2}-1)\times(\frac{2}{\sqrt{2}})^2 + (1-\frac{1}{\sqrt{2}})\times(\frac{1}{2})^2 + (1-\frac{1}{\sqrt{2}})\times(\frac{1}{2})^2 \approx 0.25 \tag{2.37}$$

Hence, approximately, an upper bound of geometric error $\sigma^2_{geom} \leq 0.25$.