2 Scan Converting Circles

3 Scan Converting Circles

- SRGP does not offer a circle primitive
- Circular ellipse arc has an 8-ways symmetry
- $x^2 + y^2 = R^2$
- Circle can be translated to origin
- Draw one quarter by incrementing unit steps in x from 0 to R. Solve for +y in each step. Then exploit symmetry.
- As x approaches R the gap in value computed becomes large.
- Inefficient because of square and root calculation
- Another inefficient method is to compute (RcosΘ, RsinΘ) with Θ from 0^0 to 90^0

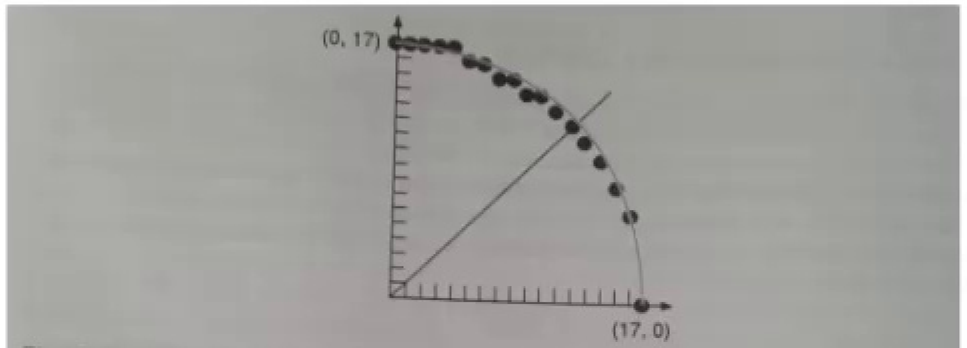
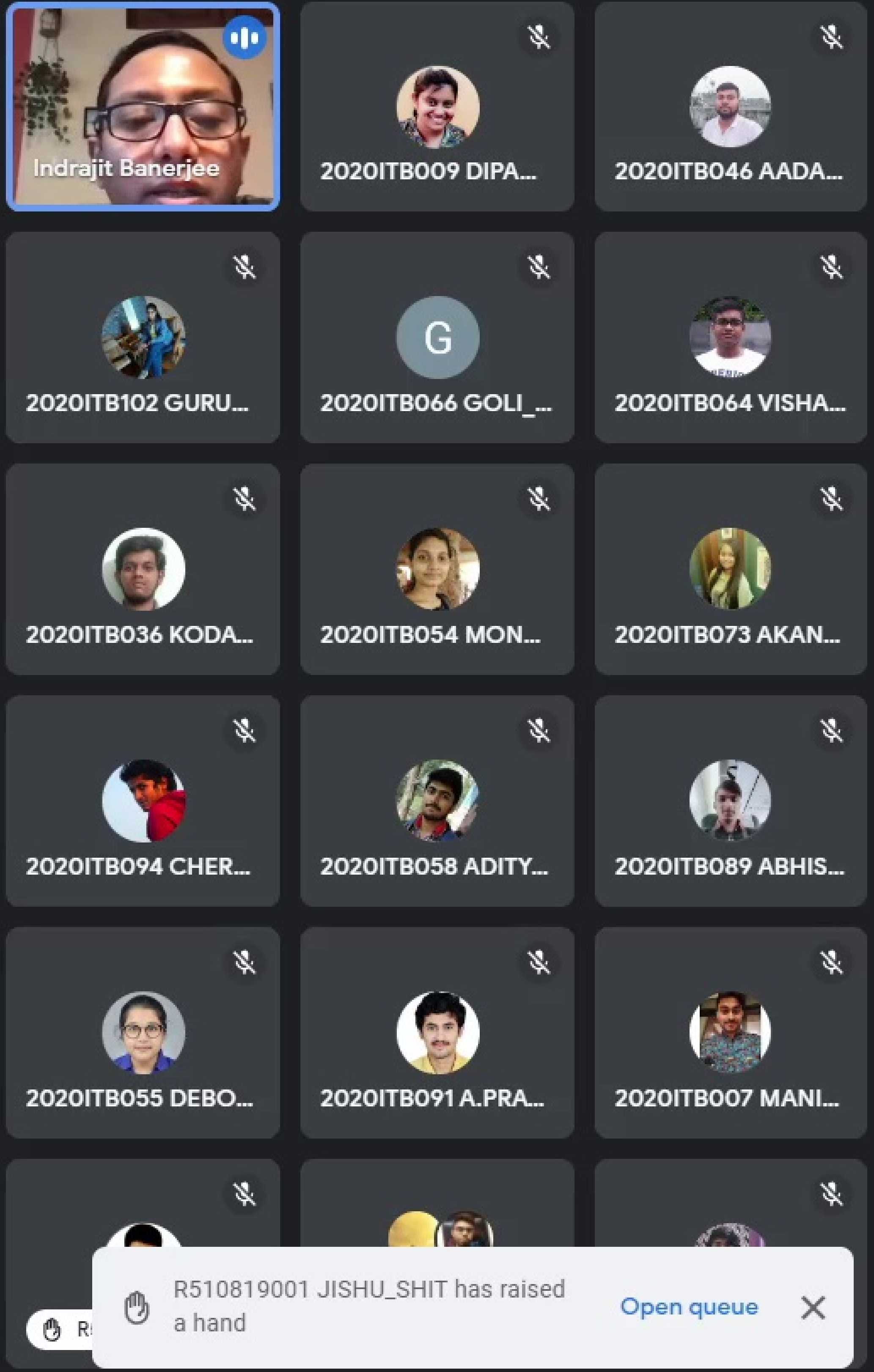


Fig. 3.12 A quarter circle generated with unit steps in x, and with y calculated and then rounded. Unique values of y for each x produce gaps.

1. Eight way symmetry
2. Midpoint circle algorithm
   1. Second Order difference



Meet - Computer Graphics T

meet.google.com/cts-ufwc-def?authuser=1

Indrajit Banerjee is presenting

2 Midpoint circle algorithm

$d_{old} = F(x_p+1,\ y_p - ½)$

$= (x_p+1)^2 + (y_p - ½)^2 + R^2$

If $d_{old} < 0$

E $d_{new} = F(x_p+2,\ y_p - ½)$

$= (x_p+2)^2 + (y_p - ½)^2 + R^2$

$= d_{old} + (2x_p + 3).\ \Delta_E = (2x_p + 3)$

If $d_{old} >= 0$

SE $d_{new} = F(x_p+2,\ y_p - 3/2)$

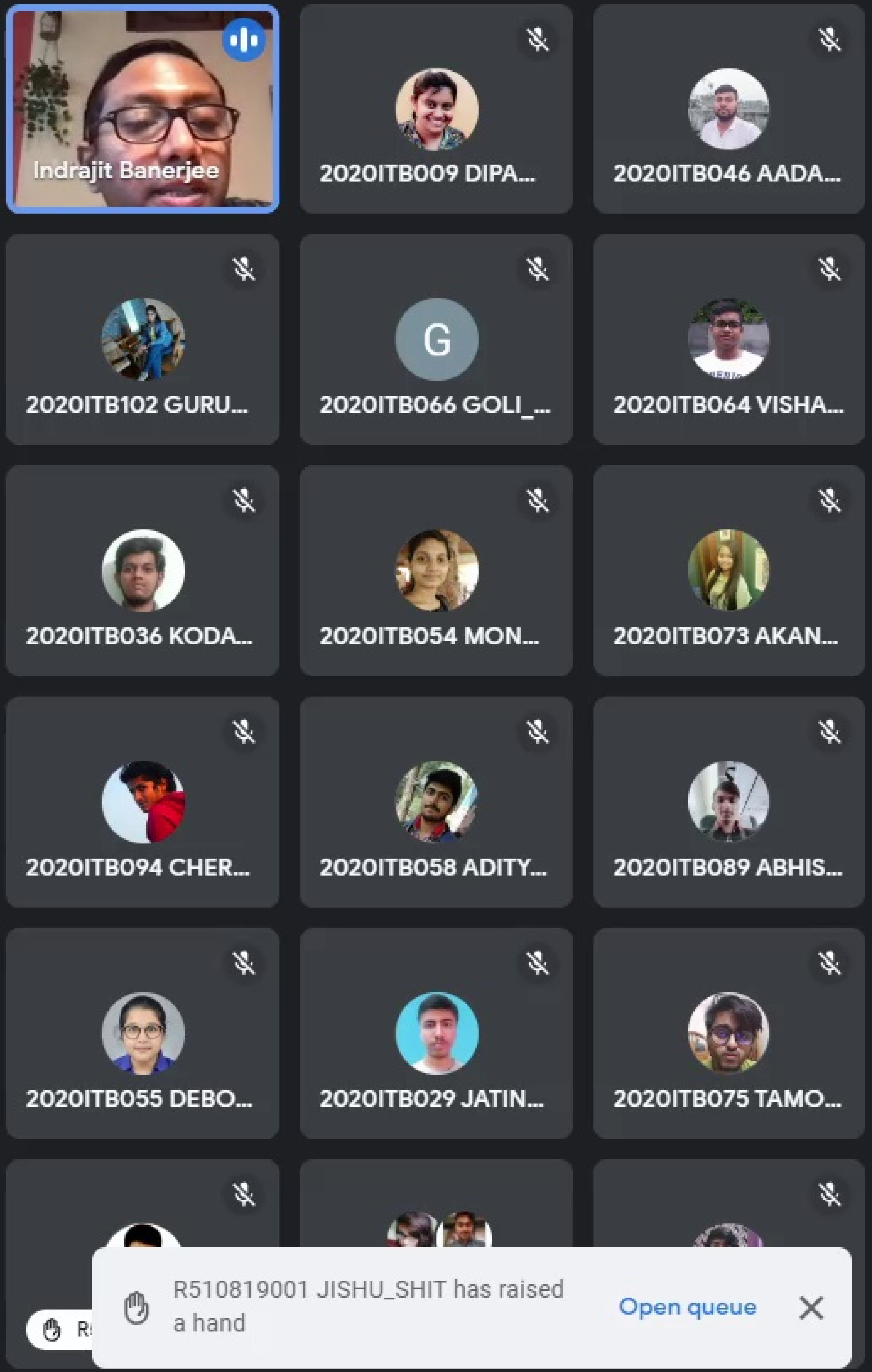
$= (x_p+2)^2 + (y_p - 3/2)^2 + R^2$

$= d_{old} + (2x_p - 2y_p + 5).\ \Delta_{SE} = 2x_p - 2y_p + 5$

- Δ_E and Δ_{SE} Vary at each step unlike the linear case.
- (x_p, y_p) is called the point of evaluation
- Chose the pixel based on the sign of variable d in the previous iteration
- Update the decision variable d with the Δ that corresponds to the choice of pixel.
- Starting point is (0, R).
- The next mid-point is (1, R- ½)
- $F(1, R- ½) = 1 + (R^2 +- R + ¼) - R^2 = 5/4 - R$

```
void MidpointCircle (int radius, int value)
/* Assumes center of circle is at origin */
{
    int x = 0;
    int y = radius;
    double d = 5.0 / 4.0 - radius;
    CirclePoints (x, y, value);

    while (y > x) {
        if (d < 0)          /* Select E */
            d += 2.0 * x + 3.0;
        else {              /* Select SE */
            d += 2.0 * (x - y) + 5.0;
            y--;
        }
        x++;
        CirclePoints (x, y, value);
    } /* while */
} /* MidpointCircle */
```

Fig. 3.15 The midpoint circle scan-conversion algorithm.



2020ITB018 BIGHNA_BINASHAN has left the meeting

R510819001 JISHU_SHIT has raised a hand Open queue

10:10 AM | Computer Graphics Theory

60

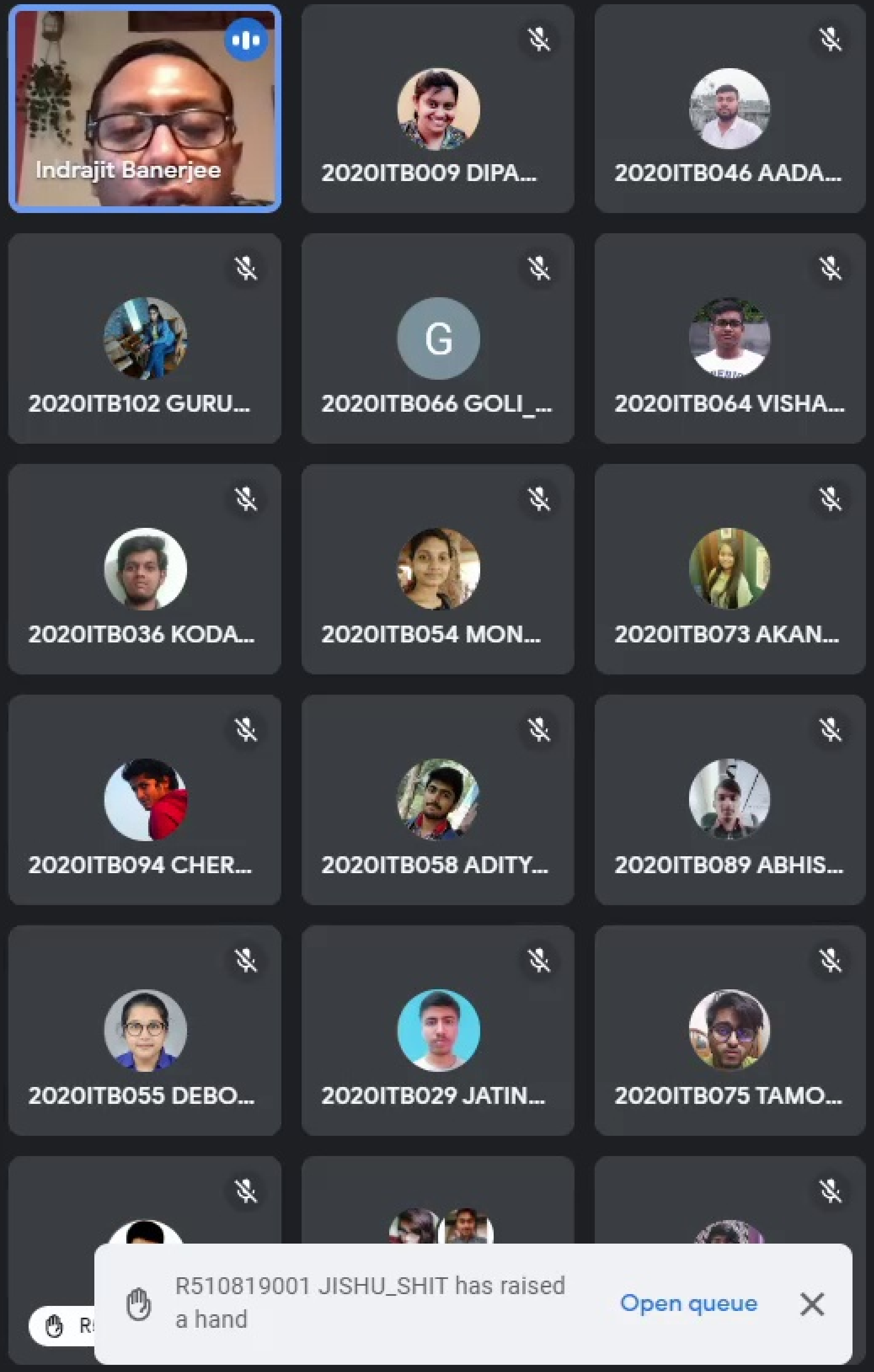
Meet - Computer Graphics T

meet.google.com/cts-ufwc-def?authuser=1

Indrajit Banerjee is presenting

Midpoint circle algorithm
Second Order Difference

- We noted that Δ functions were linear and we computed them directly.
- Any polynomial can be computed incrementally as we did with line and circle.
- We calculate the first and second order partial difference.
  - Evaluate function directly at two adjacent points
  - Calculate the difference (which for a polynomial is always a polynomial of lower degree)
  - Apply the difference in each iteration



10:12 AM | Computer Graphics Theory

61

Midpoint circle algorithm
Second Order Difference

- $\Delta_{E_{old}}(x_P, y_P) = 2x_p+3$
- $\Delta_{E_{new}}(x_p+1\ y_P)$

  $= 2(x_p+1)+ 3$

  $\Delta_{Enew} - \Delta_{Eold} = 2$
- $\Delta_{SE_{old}}(x_P, y_P) = 2x_p- 2x_p+5$
- $\Delta_{SE_{new}}(x_p+1, y_P)$

  $= 2(x_p+1)- 2y_p+5$

  $\Delta_{SEnew} - \Delta_{SEold} = 2$
- If we choose SE in current iteration point moves from (x_P, y_P) to $(x_p +1, y_p-1)$
- $\Delta_{E_{new}}(x_p+1, y_p-1)$

  $= 2(x_p+1)+ 3.$

  $\Delta_{Enew} - \Delta_{Eold} = 2$

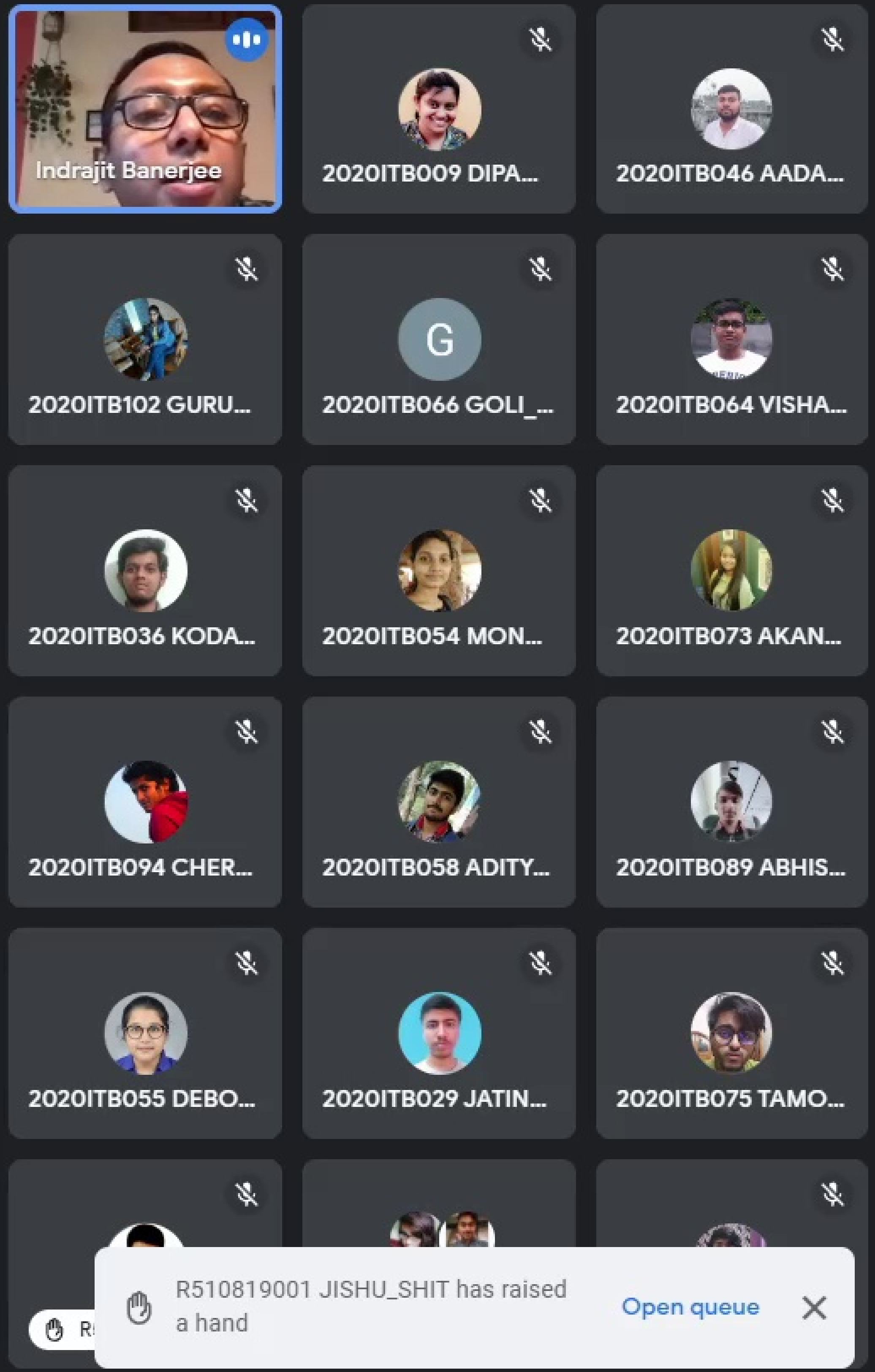
$\Delta_{SE_{new}}(x_P+1, y_p-1)$

$= 2(x_p+1)- 2(y_p-1)+5$

$\Delta_{SEnew} - \Delta_{SEold} = 4$

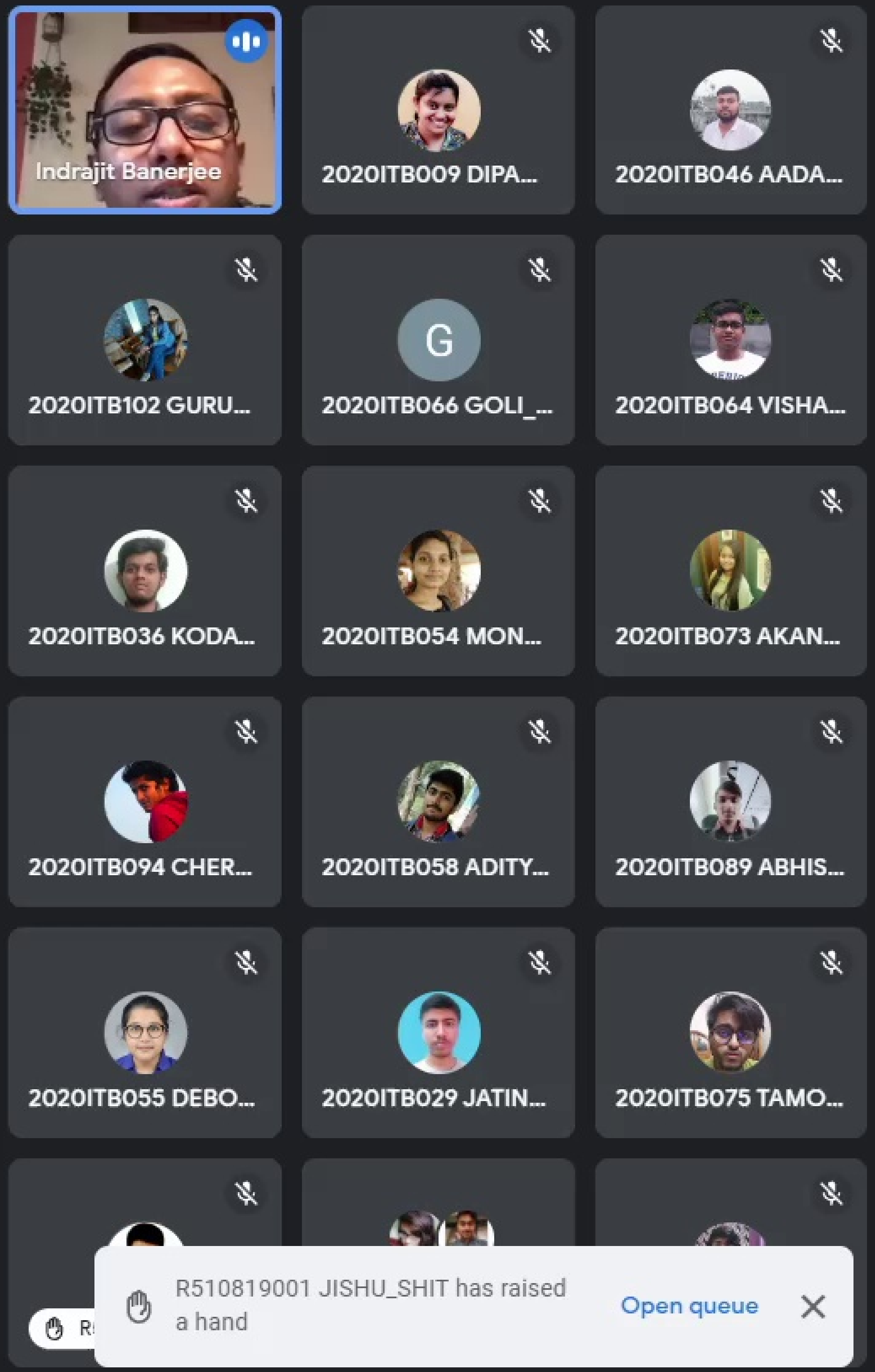
```
void MidpointCircle (int radius, int value)
/* This procedure uses second-order partial differences to compute increments */
/* in the decision variable. Assumes center of circle is at origin */
{
    int x = 0;
    int y = radius;
    int d = 1 - radius;
    int deltaE = 3;
    int deltaSE = -2 * radius + 5;
    CirclePoints (x, y, value);

    while (y > x) {
        if (d < 0) {            /* Select E */
            d += deltaE;
            deltaE += 2;
            deltaSE += 2;
        } else {
            d += deltaSE;       /* Select SE */
            deltaE += 2;
            deltaSE += 4;
            y--;
        }
        x++;
        CirclePoints (x, y, value);
    } /* while */
} /* MidpointCircle */
```

Fig. 3.18 Midpoint circle scan-conversion algorithm using second-order differences.



Midpoint circle algorithm
Second Order Difference

- We noted that Δ functions were linear and we computed them directly.
- Any polynomial can be computed incrementally as we did with line and circle.
- We calculate the first and second order partial difference.
  - Evaluate function directly at two adjacent points
  - Calculate the difference (which for a polynomial is always a polynomial of lower degree)
  - Apply the difference in each iteration



Meet - Computer Graphics T

meet.google.com/cts-ufwc-def?authuser=1

Indrajit Banerjee is presenting

2 Midpoint circle algorithm

$d_{old} = F(x_p+1, y_p - ½)$

$= (x_p+1)^2 + (y_p - ½)^{2} + R^2$

If $d_{old} < 0$

E $d_{new} = F(x_p+2, y_p - ½)$

$= (x_p+2)^2 + (y_p - ½)^{2} + R^2$

$= d_{old} + (2x_p + 3).\ \Delta_E = (2x_p + 3)$

If $d_{old} >= 0$

SE $d_{new} = F(x_p+2, y_p - 3/2)$

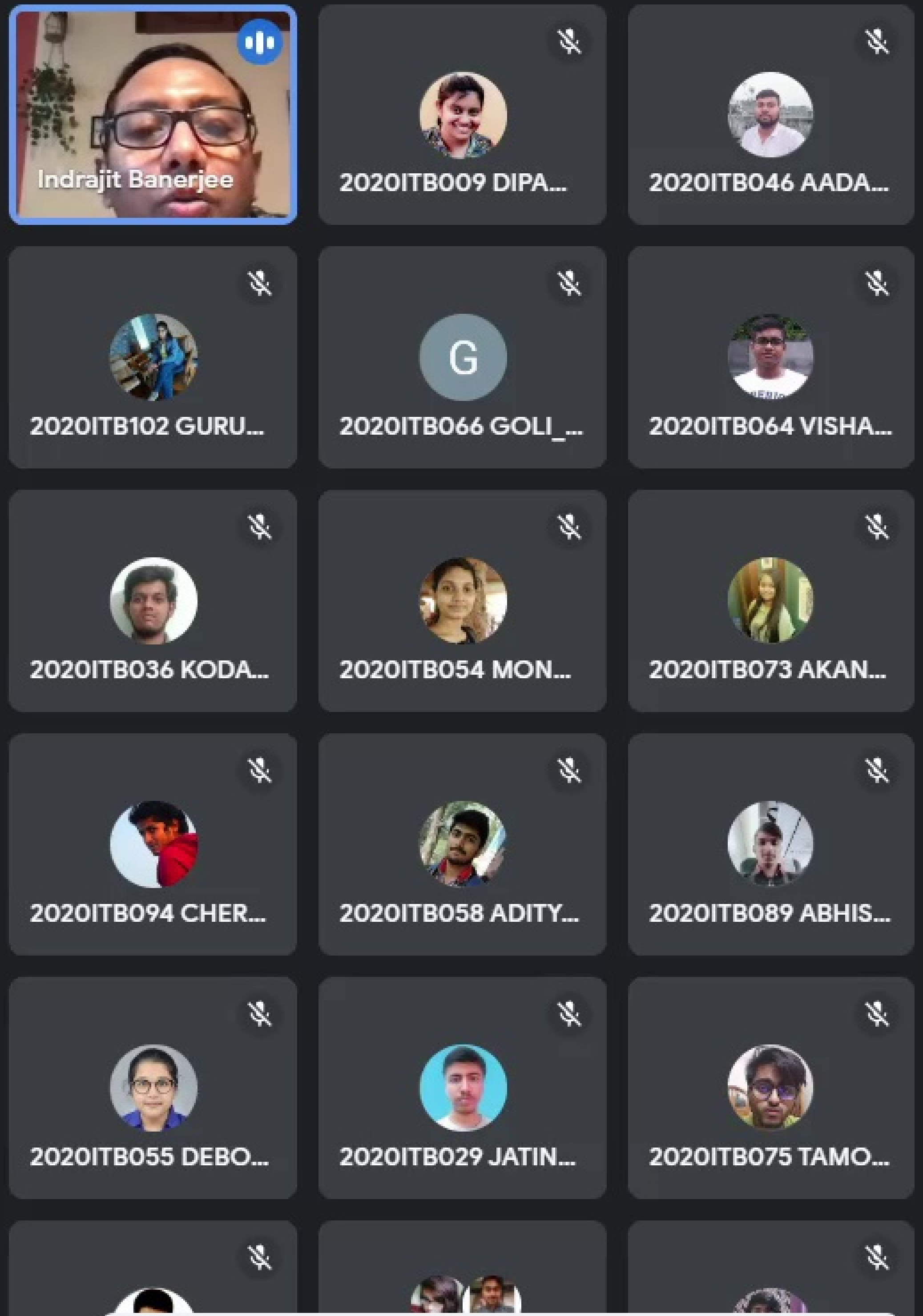
$= (x_p+2)^2 + (y_p - 3/2)^{2} + R^2$

$= d_{old} + (2x_p - 2y_p + 5).\ \Delta_{SE} = 2x_p - 2y_p + 5$

- Δ_E and Δ_{SE} Vary at each step unlike the linear case.
- (x_p, y_p) is called the point of evaluation
- Chose the pixel based on the sign of variable d in the previous iteration
- Update the decision variable d with the Δ that corresponds to the choice of pixel.
- Starting point is (0, R).
- The next mid-point is (1, R- ½)
- F (1, R- ½) = 1 +(R² +- R + ¼) - R² = 5/4 – R

```
void MidpointCircle (int radius, int value)
/* Assumes center of circle is at origin */
{
    int x = 0;
    int y = radius;
    double d = 5.0 / 4.0 - radius;
    CirclePoints (x, y, value);

    while (y > x) {
        if (d < 0)          /* Select E */
            d += 2.0 * x + 3.0;
        else {              /* Select SE */
            d += 2.0 * (x - y) + 5.0;
            y--;
        }
        x++;
        CirclePoints (x, y, value);
    } /* while */
} /* MidpointCircle */
```

Fig. 3.15 The midpoint circle scan-conversion algorithm.



R510819001 JISHU_SHIT has raised a hand

Open queue

10:14 AM | Computer Graphics Theory

63

1. Eight way symmetry

- For any given point (x,y) we can trivially draw 7 other points as in diagram.
- If we draw a 45^0 segment we can draw a circle
- Also avoid the boundary condition where x = y= R √2. This one to be drawn only 4 times.

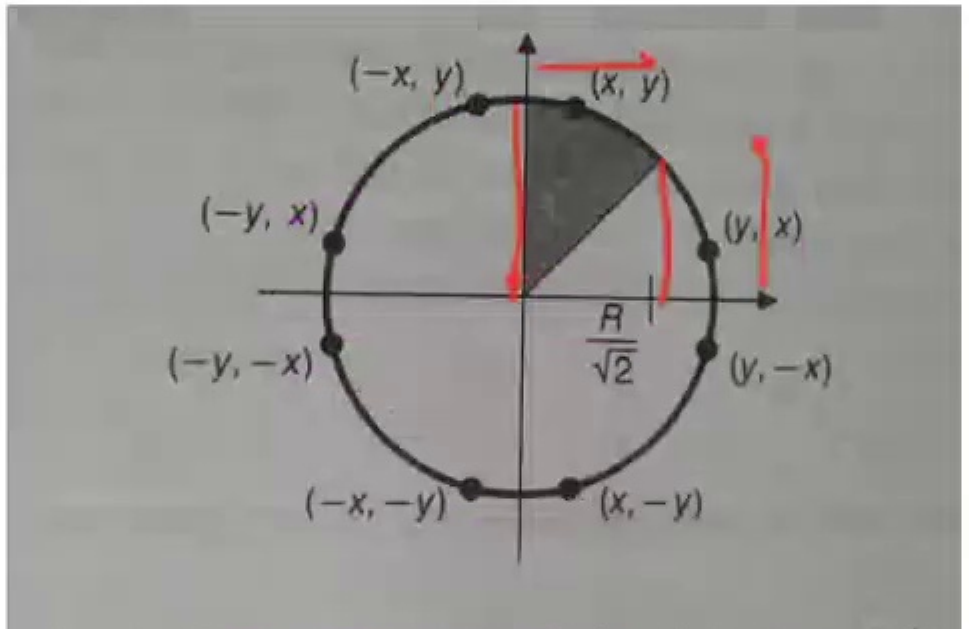


Fig. 3.13 Eight symmetrical points on a circle.

```
Void CirclePoints(int x, int y, int value)
{
        WritePixel(  x,   y,value);
        WritePixel(  y,   x,value);
        WritePixel(  y, -x,value);
        WritePixel(  x, -y,value);
        WritePixel(-x, -y,value);
        WritePixel(-y, -x,value);
        WritePixel(-y,    x,value);
        WritePixel(-x,    y,value);
        /* CirclePoints */
```

3 Scan Converting Circles

- SRGP does not offer a circle primitive
- Circular ellipse arc has an 8-ways symmetry
- $x^2 + y^2 = R^2$
- Circle can be translated to origin
- Draw one quarter by incrementing unit steps in x from 0 to R. Solve for +y in each step. Then exploit symmetry.
- As x approaches R the gap in value computed becomes large.
- Inefficient because of square and root calculation
- Another inefficient method is to compute (RcosΘ, RsinΘ) with Θ from 0^0 to 90^0

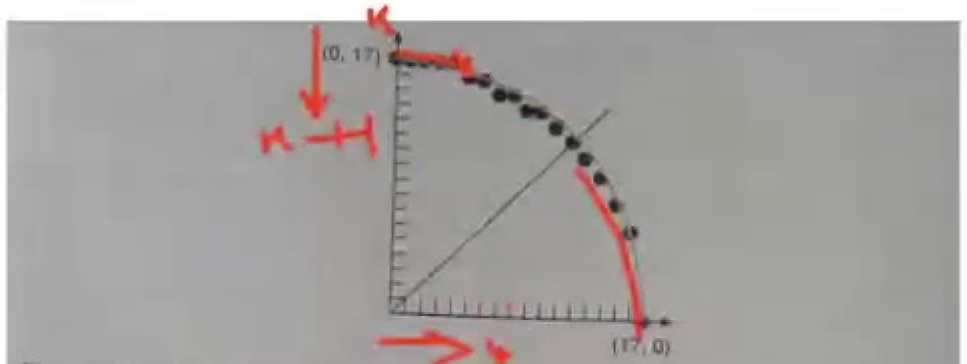


Fig. 3.12 A quarter circle generated with unit steps in x, and with y calculated and then rounded. Unique values of y for each x produce gaps.

1. Eight way symmetry
2. Midpoint circle algorithm
    1. Second Order difference

2 Midpoint circle algorithm

- Bresenham: More efficient that 8-way symmetry. Go all the way round the circle.
- Mid-point algorithm: integer centre and radius.
- Second Octant: Consider 45^0 between x=0 to x=y=R/√2.
- Evaluate mid-point between pixels and find which is closer to the circle.

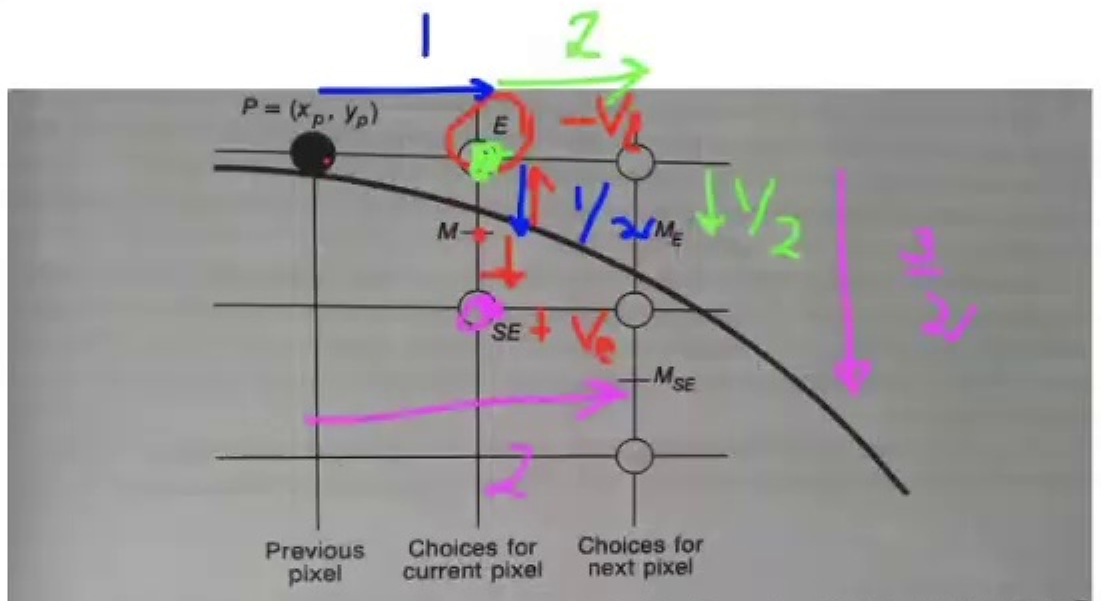


Fig. 3.14 The pixel grid for the midpoint circle algorithm showing *M* and the pixels *E* and *SE* to choose between.

If pixel P is at (x_P, y_P) has been choosen previously select between E and SE. $F(x,y) = X^2 + y^2 - R^2$ is 0 on the circle +ve outside and –ve inside. If mid-point outside select SE else E

Meet - Computer Graphics T

meet.google.com/cts-ufwc-def?authuser=1

Indrajit Banerjee is presenting

2 Midpoint circle algorithm

$d_{old} = F(x_p+1,\ y_p - ½)$
$= (x_p+1)^2 + (y_p - ½)^2 + R^2$

If $d_{old} < 0$

E $d_{new} = F(x_p+2,\ y_p - ½)$
$= (x_p+2)^2 + (y_p - ½)^2 + R^2$
$= d_{old} + (2x_p + 3).\ \Delta_E = (2x_p + 3)$

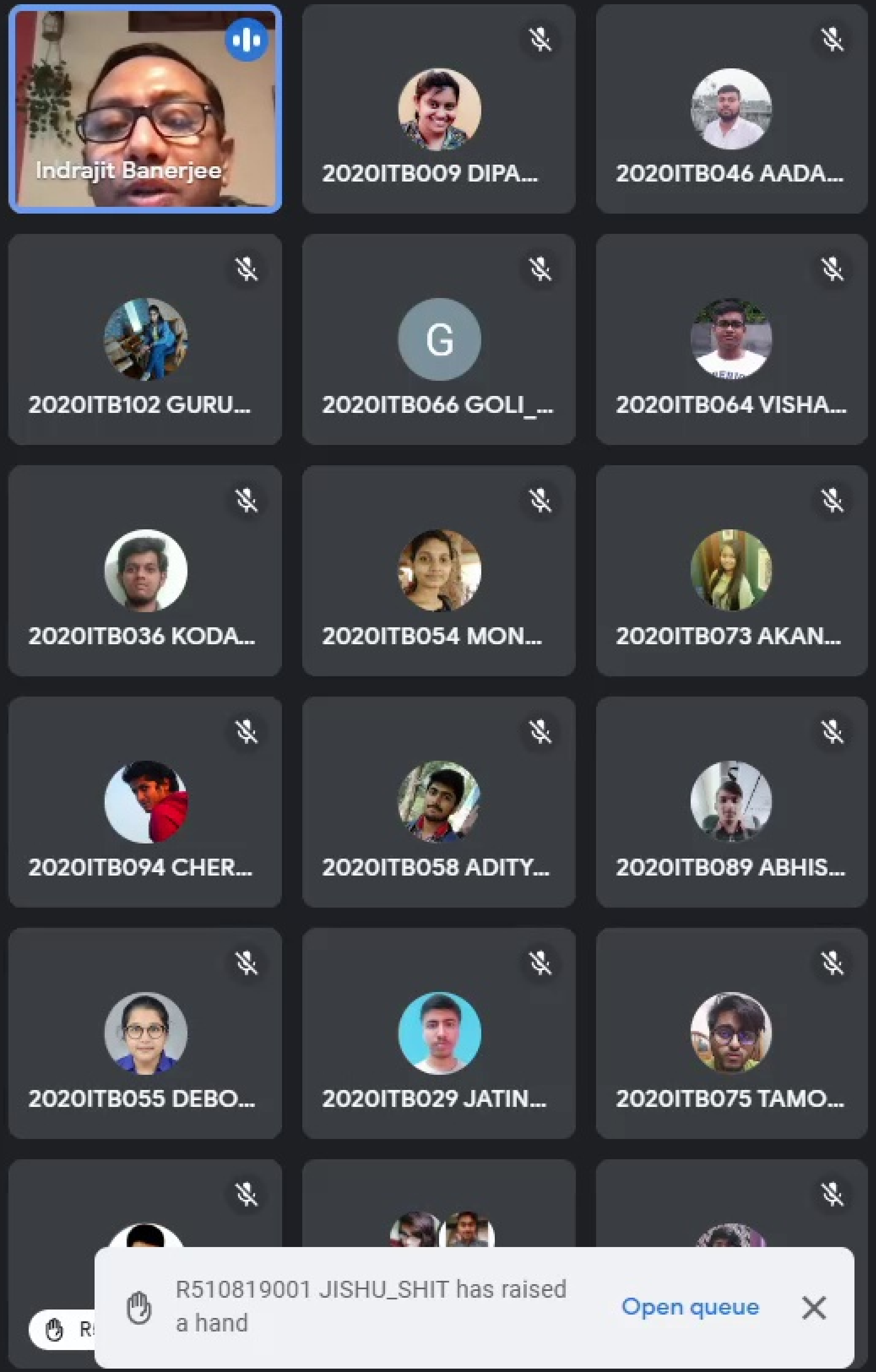
If $d_{old} >= 0$

SE $d_{new} = F(x_p+2,\ y_p - 3/2)$
$= (x_p+2)^2 + (y_p - 3/2)^2 + R^2$
$= d_{old} + (2x_p - 2y_p + 5).\ \Delta_{SE} = 2x_p - 2y_p + 5$

- Δ_E and Δ_{SE} Vary at each step unlike the linear case.
- (x_p, y_p) is called the point of evaluation
- Chose the pixel based on the sign of variable d in the previous iteration
- Update the decision variable d with the Δ that corresponds to the choice of pixel.
- Starting point is (0, R).
- The next mid-point is (1, R- ½)
- $F(1,\ R- ½) = 1 + (R^2 +- R + ¼) - R^2 = 5/4 - R$

```
void MidpointCircle (int radius, int value)
/* Assumes center of circle is at origin */
{
    int x = 0;
    int y = radius;
    double d = 5.0 / 4.0 - radius;
    CirclePoints (x, y, value);

    while (y > x) {
        if (d < 0)          /* Select E */
            d += 2.0 * x + 3.0;
        else {              /* Select SE */
            d += 2.0 * (x - y) + 5.0;
            y--;
        }
        x++;
        CirclePoints (x, y, value);
    } /* while */
} /* MidpointCircle */
```

Fig. 3.15 The midpoint circle scan-conversion algorithm.



R510819001 JISHU_SHIT has raised a hand

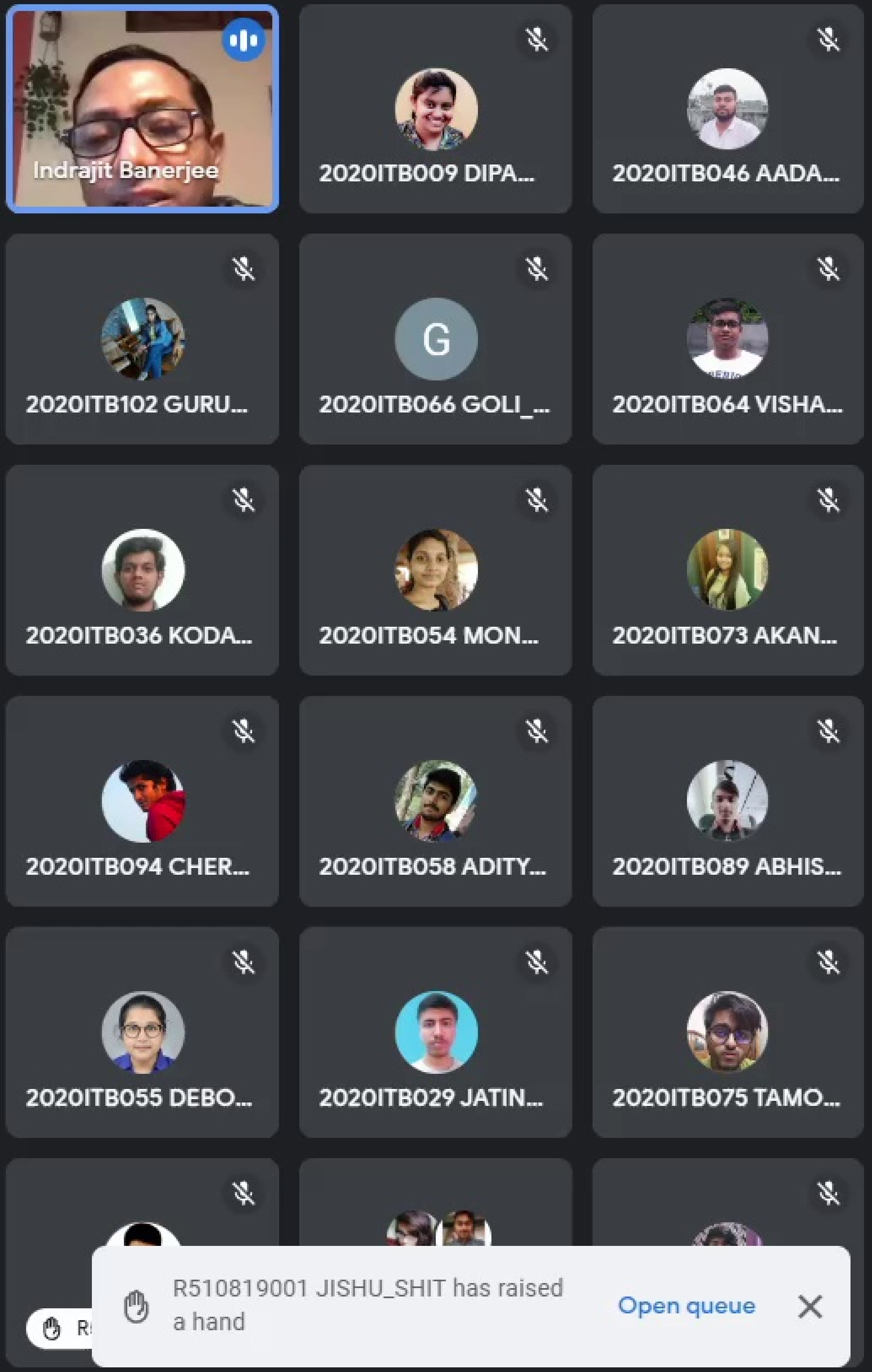
Open queue

10:15 AM | Computer Graphics Theory

63

Midpoint circle algorithm Second Order Difference

- We noted that Δ functions were linear and we computed them directly.
- Any polynomial can be computed incrementally as we did with line and circle.
- We calculate the first and second order partial difference.
  - Evaluate function directly at two adjacent points
  - Calculate the difference (which for a polynomial is always a polynomial of lower degree)
  - Apply the difference in each iteration



1. Eight way symmetry

- For any given point (x,y) we can trivially draw 7 other points as in diagram.
- If we draw a 45^0 segment we can draw a circle
- Also avoid the boundary condition where x = y= R √2. This one to be drawn only 4 times.

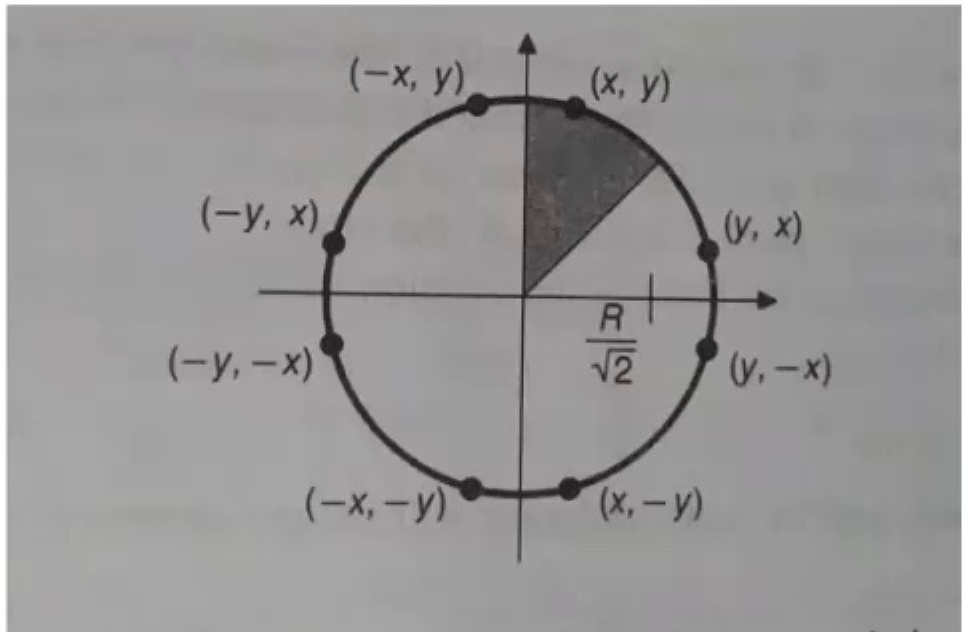


Fig. 3.13 Eight symmetrical points on a circle.

```
Void CirclePoints(int x, int y, int value)
{
        WritePixel(  x,   y,value);
        WritePixel(  y,   x,value);
        WritePixel(  y, -x,value);
        WritePixel(  x, -y,value);
        WritePixel(-x, -y,value);
        WritePixel(-y, -x,value);
        WritePixel(-y,   x,value);
        WritePixel(-x,   y,value);
        /* CirclePoints */
```

Midpoint circle algorithm Second Order Difference

- $\Delta_{E_{old}}(x_P, y_P) = 2x_P+3$
- $\Delta_{E_{new}}(x_P+1\ y_P)$

  $= 2(x_P+1)+ 3$

  $\Delta_{Enew} - \Delta_{Eold} = 2$
- $\Delta_{SE_{old}}(x_P, y_P) = 2x_P- 2x_P+5$
- $\Delta_{SE_{new}}(x_P+1, y_P)$

  $= 2(x_P+1)- 2y_P+5$

  $\Delta_{SEnew} - \Delta_{SEold} = 2$
- If we choose SE in current iteration point moves from (x_P, y_P) to (x_P+1, y_P-1)
- $\Delta_{E_{new}}(x_P+1, y_P-1)$

  $= 2(x_P+1)+ 3.$

  $\Delta_{Enew} - \Delta_{Eold} = 2$

$\Delta_{SE_{new}}(x_P+1, y_P-1)$

$= 2(x_P+1)- 2(y_P-1)+5$

$\Delta_{SEnew} - \Delta_{SEold} = 4$

```
void MidpointCircle (int radius, int value)
/* This procedure uses second-order partial differences to compute increments */
/* in the decision variable. Assumes center of circle is at origin */
{
    int x = 0;
    int y = radius;
    int d = 1 - radius;
    int deltaE = 3;
    int deltaSE = -2 * radius + 5;
    CirclePoints (x, y, value);

    while (y > x) {
        if (d < 0) {            /* Select E */
            d += deltaE;
            deltaE += 2;
            deltaSE += 2;
        } else {                /* Select SE */
            d += deltaSE;
            deltaE += 2;
            deltaSE += 4;
            y--;
        }
        x++;
        CirclePoints (x, y, value);
    }   /* while */
}   /* MidpointCircle */
```

Fig. 3.18 Midpoint circle scan-conversion algorithm using second-order differences.

4 Scan Converting Ellipses

Meet - Computer Graphics T

meet.google.com/cts-ufwc-def?authuser=1

Indrajit Banerjee is presenting

4 Scan Converting Ellipses

1. Standard ellipse centered at (0,0) is

   $b^2x^2 + a^2y^2 - a^2b^2 = 0$
2. Draw for Quad 1 and apply symmetry.
3. Divide Quad in 2 by -1 slope point.
4. The perpendicular to tangent will have slope = 1

grad $F(x,y) = \delta F/\delta x\ i + \delta F/\delta y\ j$

$= 2\,b^2x\ i + 2\,a^2y\ j$

5. $b^2x = a^2y$ and at the next mid-point $b^2(x_p+1) >= a^2(y_p - ½)$

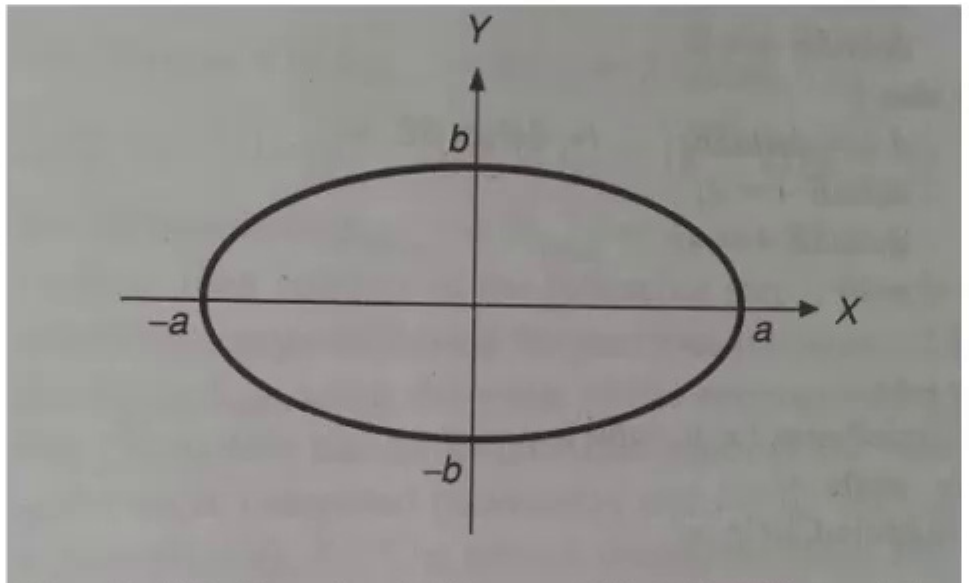
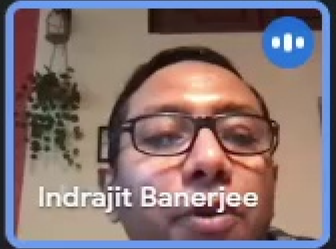


Fig. 3.19 Standard ellipse centered at the origin.



2020ITB009 DIPA...
2020ITB046 AADA...
2020ITB102 GURU...
2020ITB066 GOLI_...
2020ITB064 VISHA...
2020ITB036 KODA...
2020ITB054 MON...
2020ITB073 AKAN...
2020ITB094 CHER...
2020ITB058 ADITY...
2020ITB089 ABHIS...
2020ITB055 DEBO...
2020ITB029 JATIN...
2020ITB075 TAMO...

R510819001 JISHU_SHIT has raised a hand

Open queue

10:15 AM | Computer Graphics Theory

63

4 Scan Converting Ellipses

1. In Region 1 the y component is larger.
2. In region 1 you selct between E and SE
3. In region 2 select between S and SE

$d_{old}=F(x_p+1, y_p - ½) = b^2(x_p+1)^2 + a^2(y_p - ½)^2 - a^2b^2$

$d_{new}=F(x_p+2, y_p - ½) = b^2(x_p+2)^2 + a^2(y_p - ½)^2 - a^2b^2$ To move E

$d_{new}= d_{old} + b^2(2x_p+3)$

$\Delta_E = b^2(2x_p+3)$

$d_{new}=F(x_p+2, y_p - 3/2) = b^2(x_p+2)^2 + a^2(y_p - 3/2)^2 - a^2b^2$ To move SE

$d_{new}= d_{old} + b^2(2x_p+3) + a^2(-2y_p + 2)$

$\Delta_{SE} = b^2(2x_p+3) + a^2(-2y_p + 2)$

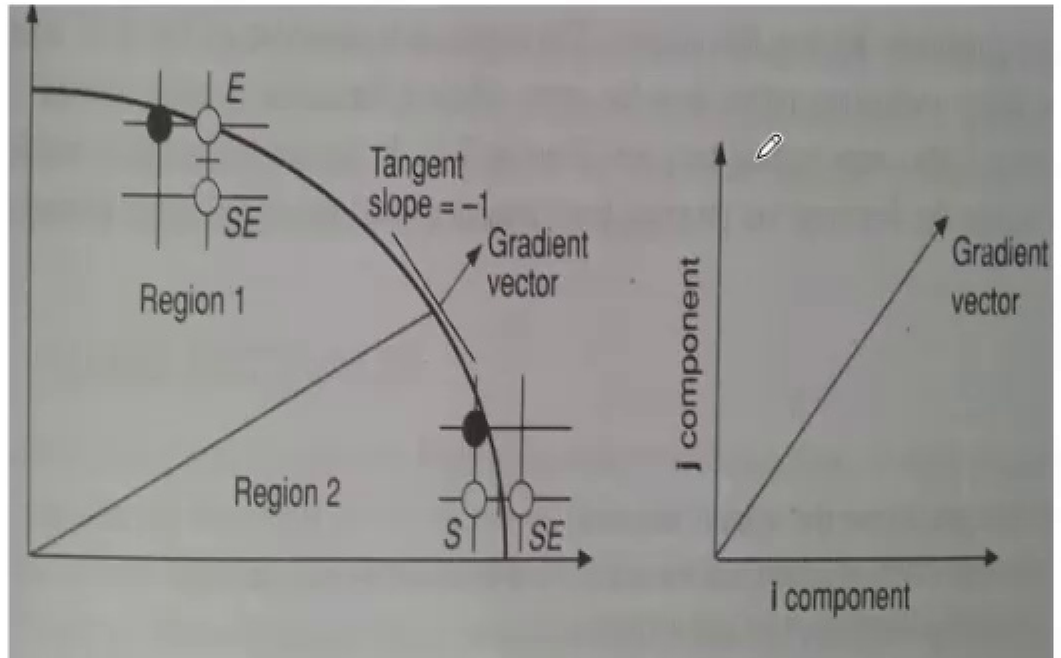
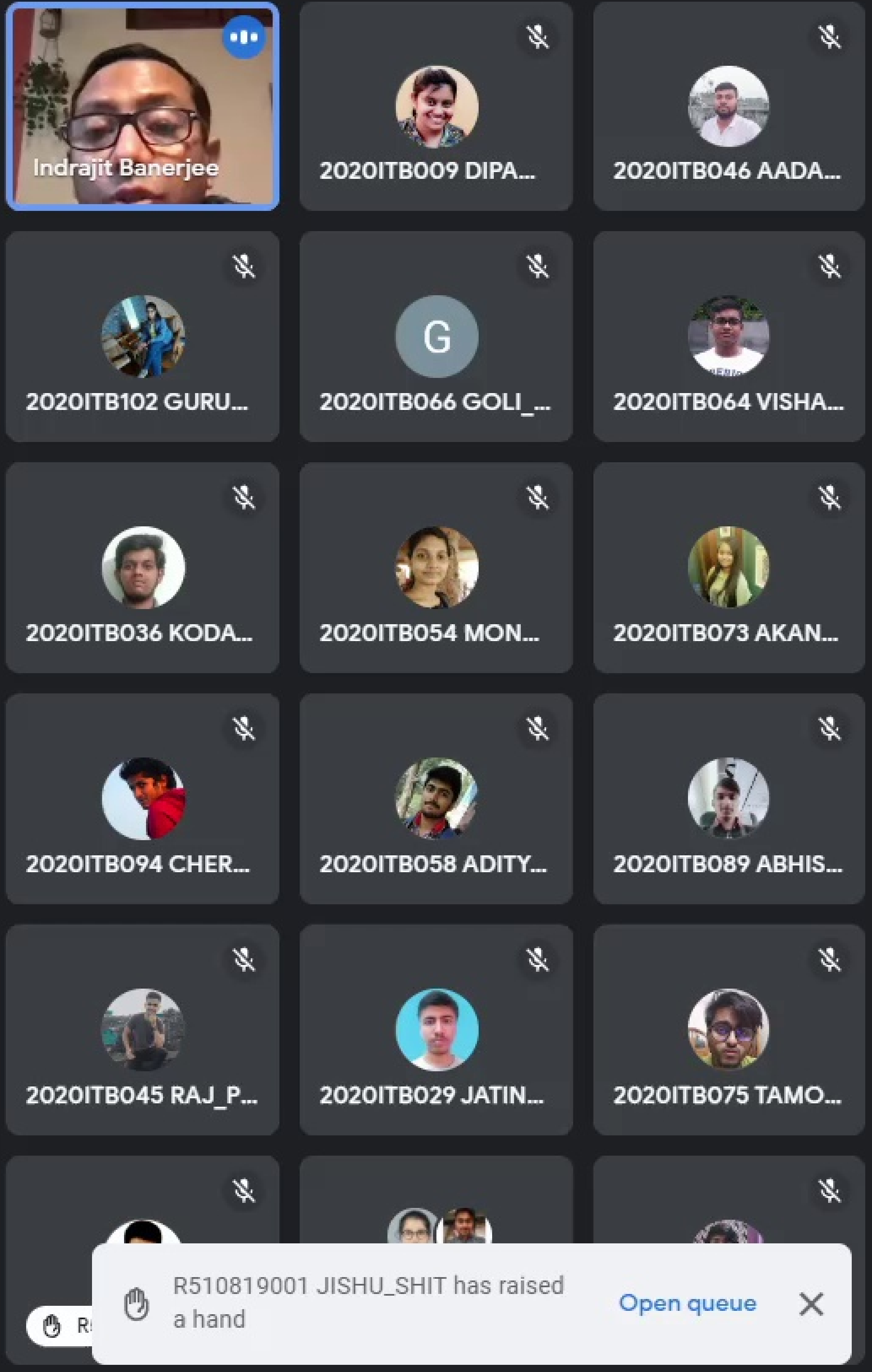


Fig. 3.20 Two regions of the ellipse defined by the 45° tangent.

4 In Region 2 decision variable d_2 is $F(x_p+ 1/2 , y_p - 1)$



4 Scan Converting Ellipses

1. Start at (0, b) and the first mid-point will be (1, b- ½) at each mid-point decide if you need to shift region by gradient evaluation.

   F(1, b – 1/2) = $b^2 + a^2 (b – 1/2)^2 - a^2b^2 = b^2 + a^2(-b +1/4)$.

```
void MidpointEllipse (int a, int b, int value)
/* Assumes center of ellipse is at the origin. Note that overflow may occur */
/* for 16-bit integers because of the squares. */
{
    double d2;

    int x = 0;
    int y = b;
    double d1 = b² – (a²b) + (0.25 a²);
    EllipsePoints (x, y, value);                    /* The 4-way symmetrical WritePixel */

    /* Test gradient if still in region 1 */
    while ( a²(y – 0.5) > b²(x + 1) ) {             /* Region 1 */
        if (d1 < 0)                                  /* Select E */
            d1 += b²(2x + 3);
        else {                                       /* Select SE */
            d1 += b²(2x + 3) + a²(–2y + 2);
            y––;
        }
        x++;
        EllipsePoints (x, y, value);
    }  /* Region 1 */

    d2 = b²(x + 0.5)² + a²(y – 1)² – a²b²;
    while (y > 0) {                                  /* Region 2 */
        if (d2 < 0) {                                /* Select SE */
            d2 += b²(2x + 2) + a²(–2y + 3);
            x++;
        } else
            d2 += a²(–2y + 3);                       /* Select S */
        y––;
        EllipsePoints (x, y, value);
    }  /* Region 2 */
}  /* MidpointEllipse */
```

Fig. 3.21 Pseudocode for midpoint ellipse scan-conversion algorithm.

4 Scan Converting Ellipses

1. In Region 1 the y component is larger.
2. In region 1 you selct between E and SE
3. In region 2 select between S and SE

$d_{old}=F(x_p+1, y_p-½) = b^2(x_p+1)^2 + a^2(y_p-½)^2 - a^2b^2$

$d_{new}=F(x_p+2, y_p-½) = b^2(x_p+2)^2 + a^2(y_p-½)^2 - a^2b^2$ To move E

$d_{new}= d_{old} + b^2(2x_p+3)$

$\Delta_E = b^2(2x_p+3)$

$d_{new}=F(x_p+2, y_p-3/2) = b^2(x_p+2)^2 + a^2(y_p-3/2)^2 - a^2b^2$ To move SE

$d_{new}= d_{old} + b^2(2x_p+3) + a^2(-2y_p+2)$

$\Delta_{SE} = b^2(2x_p+3) + a^2(-2y_p+2)$

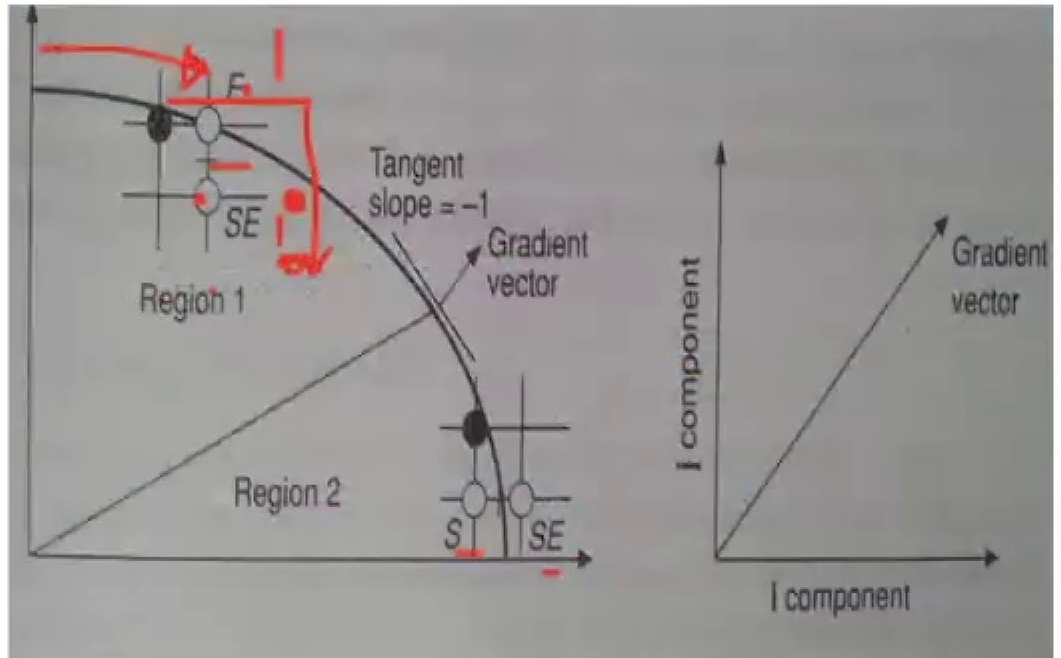
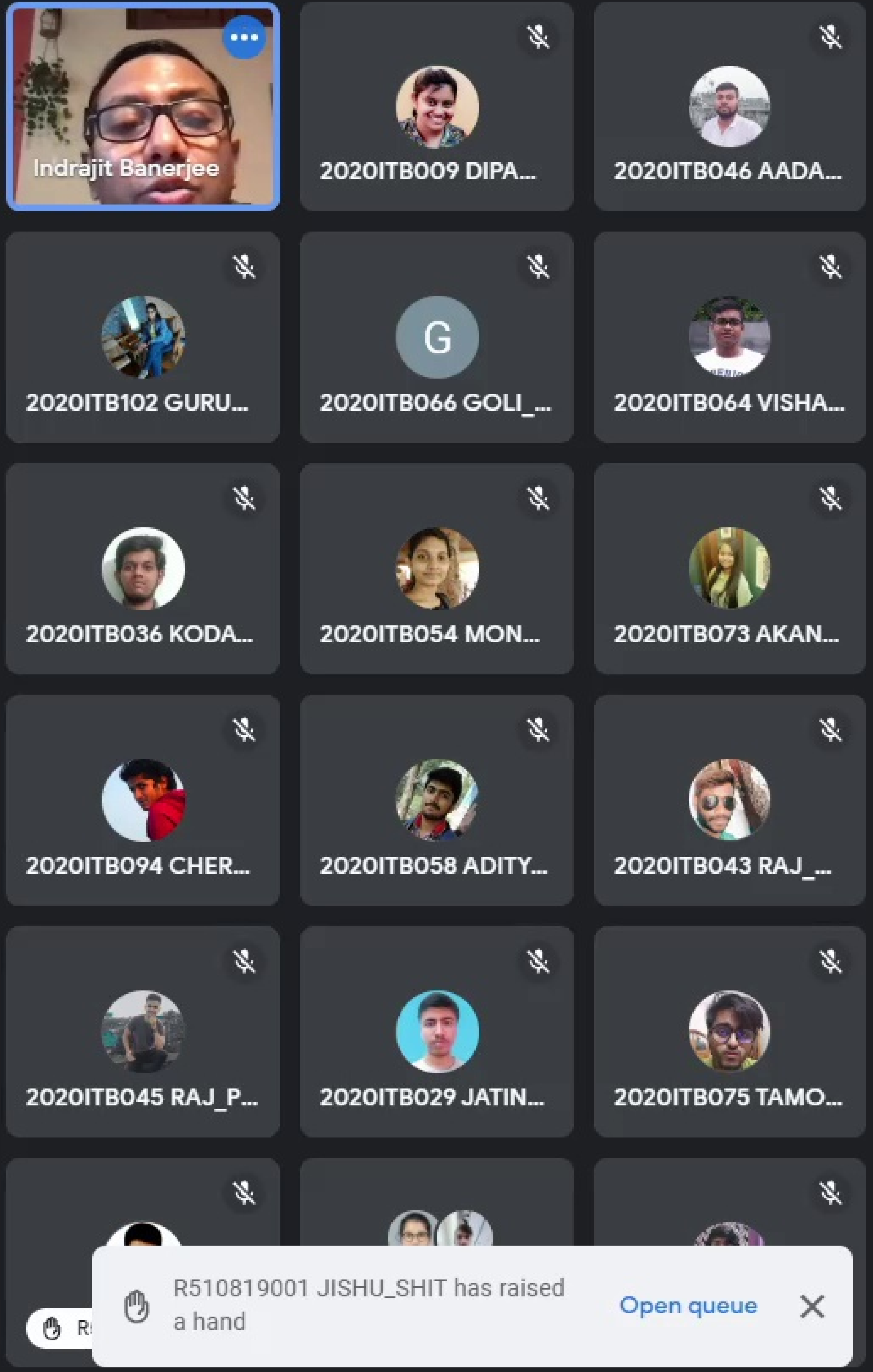


Fig. 3.20 Two regions of the ellipse defined by the 45° tangent.

4 In Region 2 decision variable d_2 is $F(x_p+1/2, y_p-1)$



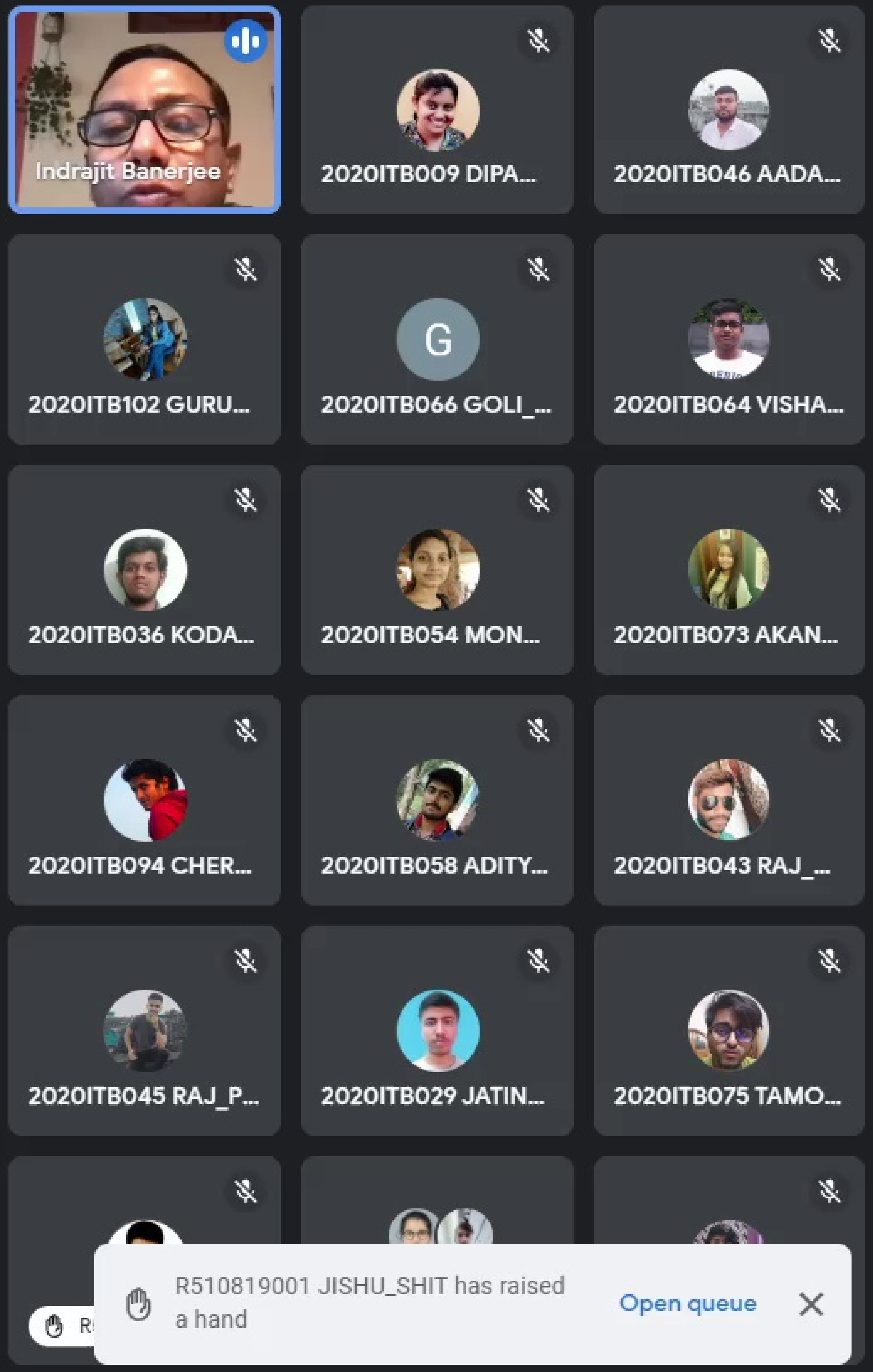
4 Scan Converting Ellipses

1. Start at (0, b) and the first mid-point will be (1, b- ½) at each mid-point decide if you need to shift region by gradient evaluation.

   $F(1, b - 1/2) = b^2 + a^2 (b - 1/2)^2 - a^2b^2 = b^2 + a^2(-b + 1/4)$.

```
void MidpointEllipse (int a, int b, int value)
/* Assumes center of ellipse is at the origin. Note that overflow may occur */
/* for 16-bit integers because of the squares. */
{
    double d2;

    int x = 0;
    int y = b;
    double d1 = b² − (a²b) + (0.25 a²);
    EllipsePoints (x, y, value);                  /* The 4-way symmetrical WritePixel */

    /* Test gradient if still in region 1 */
    while ( a²(y − 0.5) > b²(x + 1) ) {           /* Region 1 */
        if (d1 < 0)                               /* Select E */
            d1 += b²(2x + 3);
        else {                                    /* Select SE */
            d1 += b²(2x + 3) + a²(−2y + 2);
            y−−;
        }
        x++;
        EllipsePoints (x, y, value);
    } /* Region 1 */

    d2 = b²(x + 0.5)² + a²(y − 1)² − a²b²;
    while (y > 0) {                               /* Region 2 */
        if (d2 < 0) {                             /* Select SE */
            d2 += b²(2x + 2) + a²(−2y + 3);
            x++;
        } else
            d2 += a²(−2y + 3);                    /* Select S */
        y−−;
        EllipsePoints (x, y, value);
    } /* Region 2 */
/* MidpointEllipse */
```

Fig. 3.21 Pseudocode for midpoint ellipse scan-conversion algorithm.



Thank You

- End of Session 2
- All material taken from text.
- Computer Graphics
  - Principles and Practice in C
  - By James D. Foley, Andries van Dam, Steven K. Feiner, F. Hughes John
- Text Reference:
  - Chapter 3: Section 3.1 to 3.6

2 Midpoint circle algorithm

- Bresenham: More efficient that 8-way symmetry. Go all the way round the circle.
- Mid-point algorithm: integer centre and radius.
- Second Octant: Consider 45^0 between x=0 to $x=y=R/\sqrt{2}$.
- Evaluate mid-point between pixels and find which is closer to the circle.

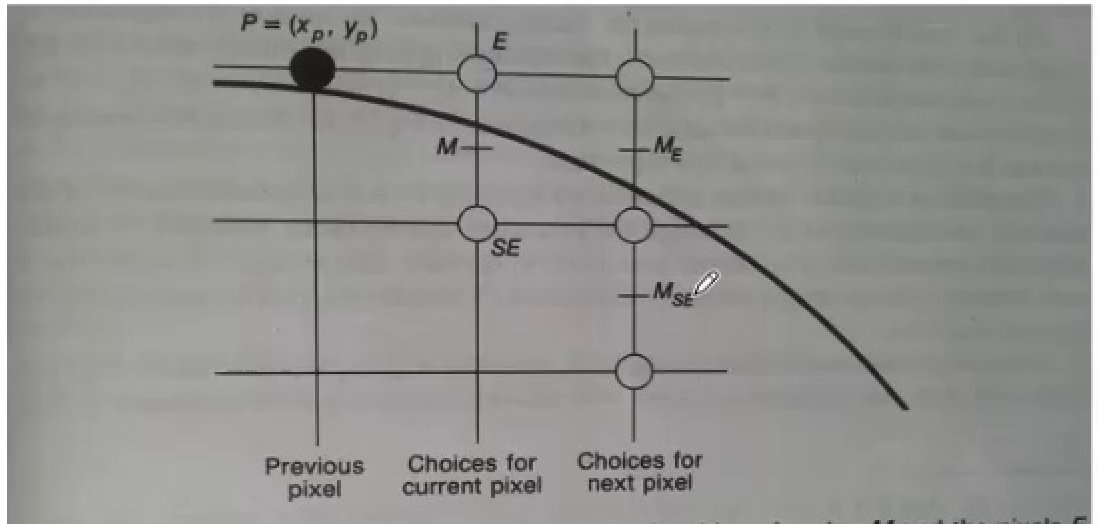


Fig. 3.14 The pixel grid for the midpoint circle algorithm showing *M* and the pixels *E* and *SE* to choose between.

If pixel P is at (x_P, y_P) has been choosen previously select between E and SE.
$F(x,y) = X^2 + y^2 - R^2$ is 0 on the circle +ve outside and –ve inside. If mid-point outside select SE else E

2 Midpoint circle algorithm

$d_{old} = F(x_P+1, y_P - ½)$

$= (x_P+1)^2 + (y_P - ½)^{2} + R^2$

If $d_{old} < 0$

E $d_{new} = F(x_P+2, y_P - ½)$

$= (x_P+2)^2 + (y_P - ½)^{2} + R^2$

$= d_{old} + (2x_P + 3)$. $\Delta_E = (2x_P + 3)$

If $d_{old} >= 0$

SE $d_{new} = F(x_P+2, y_P - 3/2)$

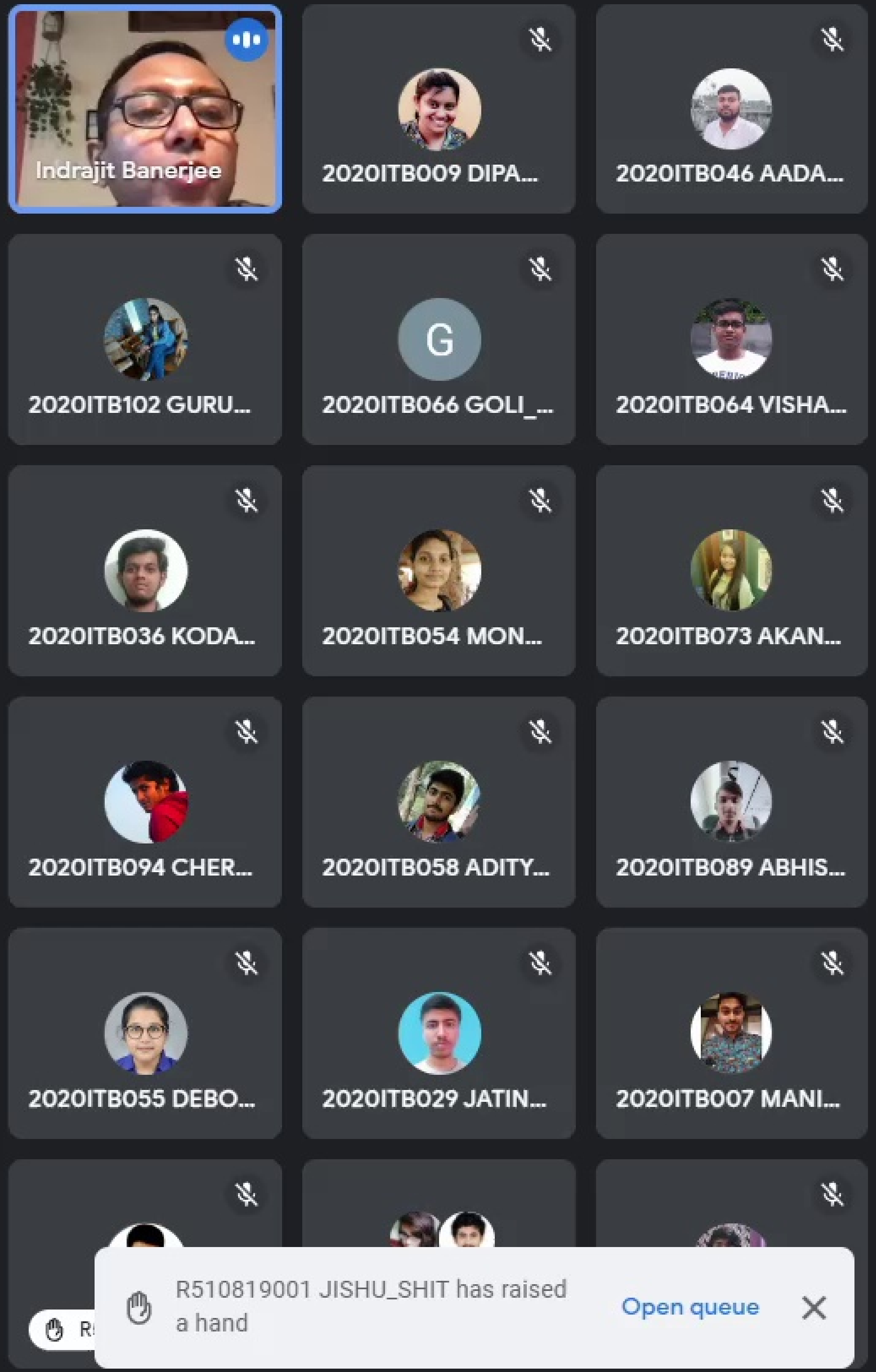
$= (x_P+2)^2 + (y_P - 3/2)^{2} + R^2$

$= d_{old} + (2x_P - 2y_P + 5)$. $\Delta_{SE} = 2x_P - 2y_P + 5$

- Δ_E and Δ_{SE} Vary at each step unlike the linear case.
- (x_P, y_P) is called the point of evaluation
- Chose the pixel based on the sign of variable d in the previous iteration
- Update the decision variable d with the Δ that corresponds to the choice of pixel.
- Starting point is (0, R).
- The next mid-point is (1, R- ½)
- F (1, R- ½) = 1 +(R² +- R + ¼) - R² = 5/4 – R

```
void MidpointCircle (int radius, int value)
/* Assumes center of circle is at origin */
{
    int x = 0;
    int y = radius;
    double d = 5.0 / 4.0 − radius;
    CirclePoints (x, y, value);

    while (y > x) {
        if (d < 0)          /* Select E */
            d += 2.0 * x + 3.0;
        else {              /* Select SE */
            d += 2.0 * (x − y) + 5.0;
            y−−;
        }
        x++;
        CirclePoints (x, y, value);
    } /* while */
} /* MidpointCircle */
```

Fig. 3.15 The midpoint circle scan-conversion algorithm.



2 Midpoint circle algorithm

- Bresenham: More efficient that 8-way symmetry. Go all the way round the circle.
- Mid-point algorithm: integer centre and radius.
- Second Octant: Consider 45^0 between x=0 to $x=y=R/\sqrt{2}$.
- Evaluate mid-point between pixels and find which is closer to the circle.

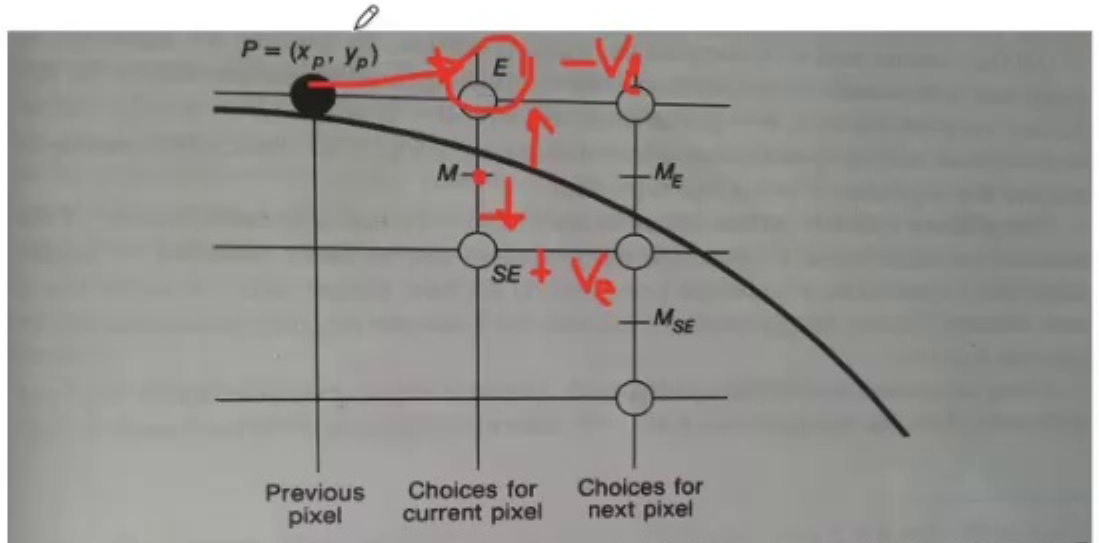


Fig. 3.14 The pixel grid for the midpoint circle algorithm showing *M* and the pixels *E* and *SE* to choose between.

If pixel P is at (x_P, y_P) has been choosen previously select between E and SE.
$F(x,y) = X^2 + y^2 - R^2$ is 0 on the circle +ve outside and –ve inside. If mid-point outside select SE else E

Meet - Computer Graphics T

meet.google.com/cts-ufwc-def?authuser=1

Indrajit Banerjee is presenting

2 Midpoint circle algorithm

$d_{old} = F(x_p+1, y_p - ½)$
$= (x_p+1)^2 + (y_p - ½)^{2} + R^2$

If $d_{old} < 0$

E $d_{new} = F(x_p+2, y_p - ½)$
$= (x_p+2)^2 + (y_p - ½)^{2} + R^2$
$= d_{old} + (2x_p + 3)$. $\Delta_E = (2x_p + 3)$

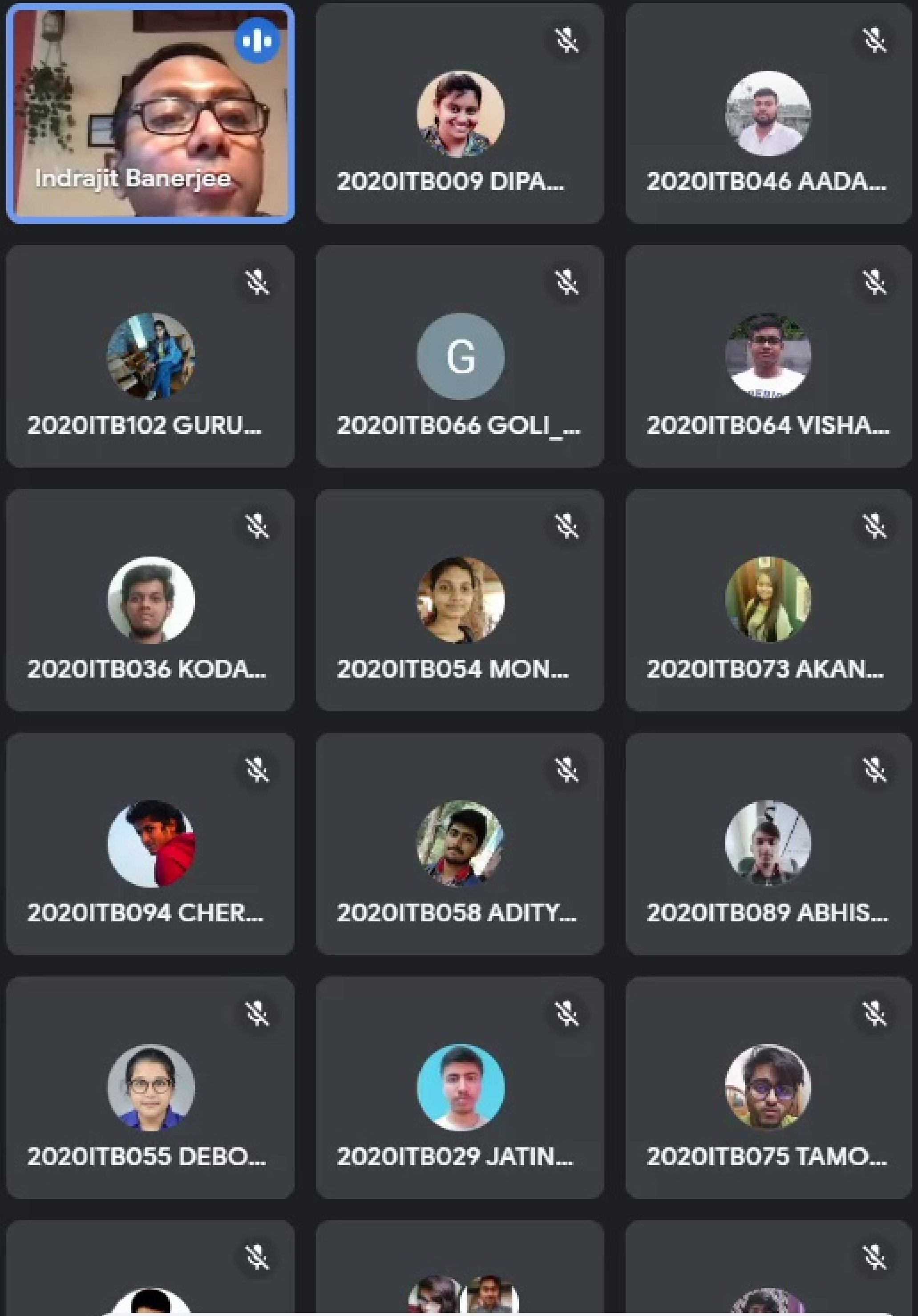
If $d_{old} >= 0$

SE $d_{new} = F(x_p+2, y_p - 3/2)$
$= (x_p+2)^2 + (y_p - 3/2)^{2} + R^2$
$= d_{old} + (2x_p - 2y_p + 5)$. $\Delta_{SE} = 2x_p - 2y_p + 5$

- Δ_E and Δ_{SE} Vary at each step unlike the linear case.
- (x_p, y_p) is called the point of evaluation
- Chose the pixel based on the sign of variable d in the previous iteration
- Update the decision variable d with the Δ that corresponds to the choice of pixel.
- Starting point is (0, R).
- The next mid-point is (1, R- ½)
- F (1, R- ½) = 1 +(R² +- R + ¼) - R² = 5/4 – R

```
void MidpointCircle (int radius, int value)
/* Assumes center of circle is at origin */
{
    int x = 0;
    int y = radius;
    double d = 5.0 / 4.0 – radius;
    CirclePoints (x, y, value);

    while (y > x) {
        if (d < 0)          /* Select E */
            d += 2.0 * x + 3.0;
        else {              /* Select SE */
            d += 2.0 * (x – y) + 5.0;
            y––;
        }
        x++;
        CirclePoints (x, y, value);
    } /* while */
} /* MidpointCircle */
```

Fig. 3.15 The midpoint circle scan-conversion algorithm.



R510819001 JISHU_SHIT has raised a hand

Open queue

59

10:08 AM | Computer Graphics Theory

2 Midpoint circle algorithm

- Bresenham: More efficient that 8-way symmetry. Go all the way round the circle.
- Mid-point algorithm: integer centre and radius.
- Second Octant: Consider 45^0 between x=0 to x=y=R/√2.
- Evaluate mid-point between pixels and find which is closer to the circle.

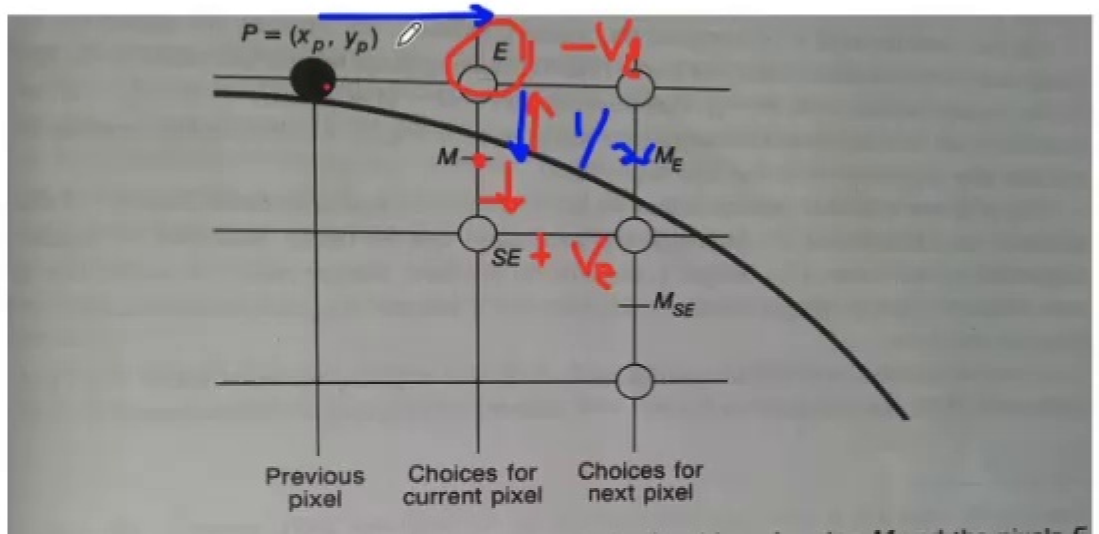
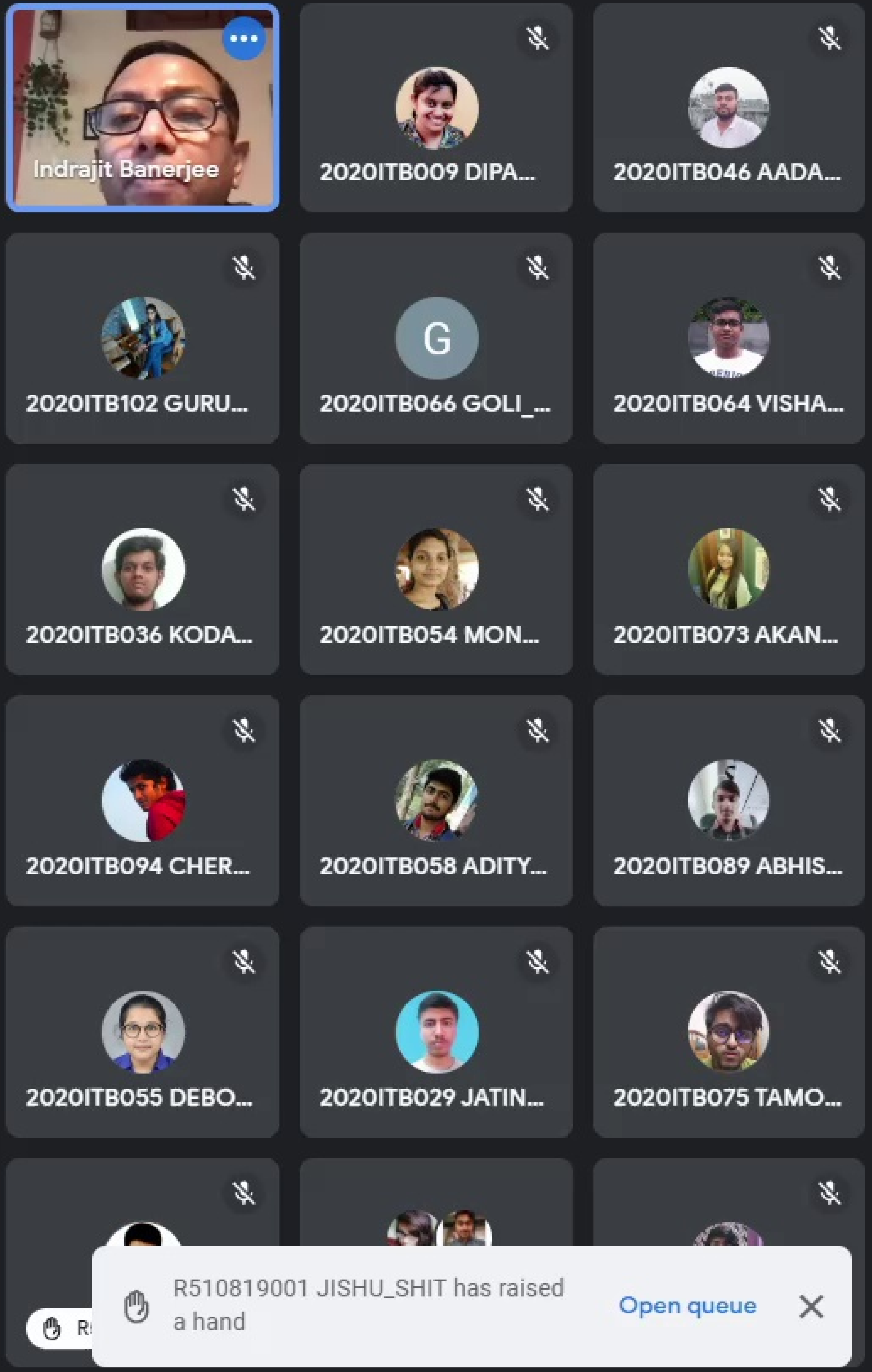


Fig. 3.14 The pixel grid for the midpoint circle algorithm showing *M* and the pixels *E* and *SE* to choose between.

If pixel P is at (x_P, y_P) has been choosen previously select between E and SE.
$F(x,y) = X^2 + y^2 - R^2$ is 0 on the circle +ve outside and –ve inside. If mid-point outside select SE else E



2 Midpoint circle algorithm

$d_{old} = F(x_P+1,\ y_P - ½)$

$= (x_P+1)^2 + (y_P - ½)^2 + R^2$

If $d_{old} < 0$

E $d_{new} = F(x_P+2,\ y_P - ½)$

$= (x_P+2)^2 + (y_P - ½)^2 + R^2$

$= d_{old} + (2x_P + 3).\ \Delta_E = (2x_P + 3)$

If $d_{old} >= 0$

SE $d_{new} = F(x_P+2,\ y_P - 3/2)$

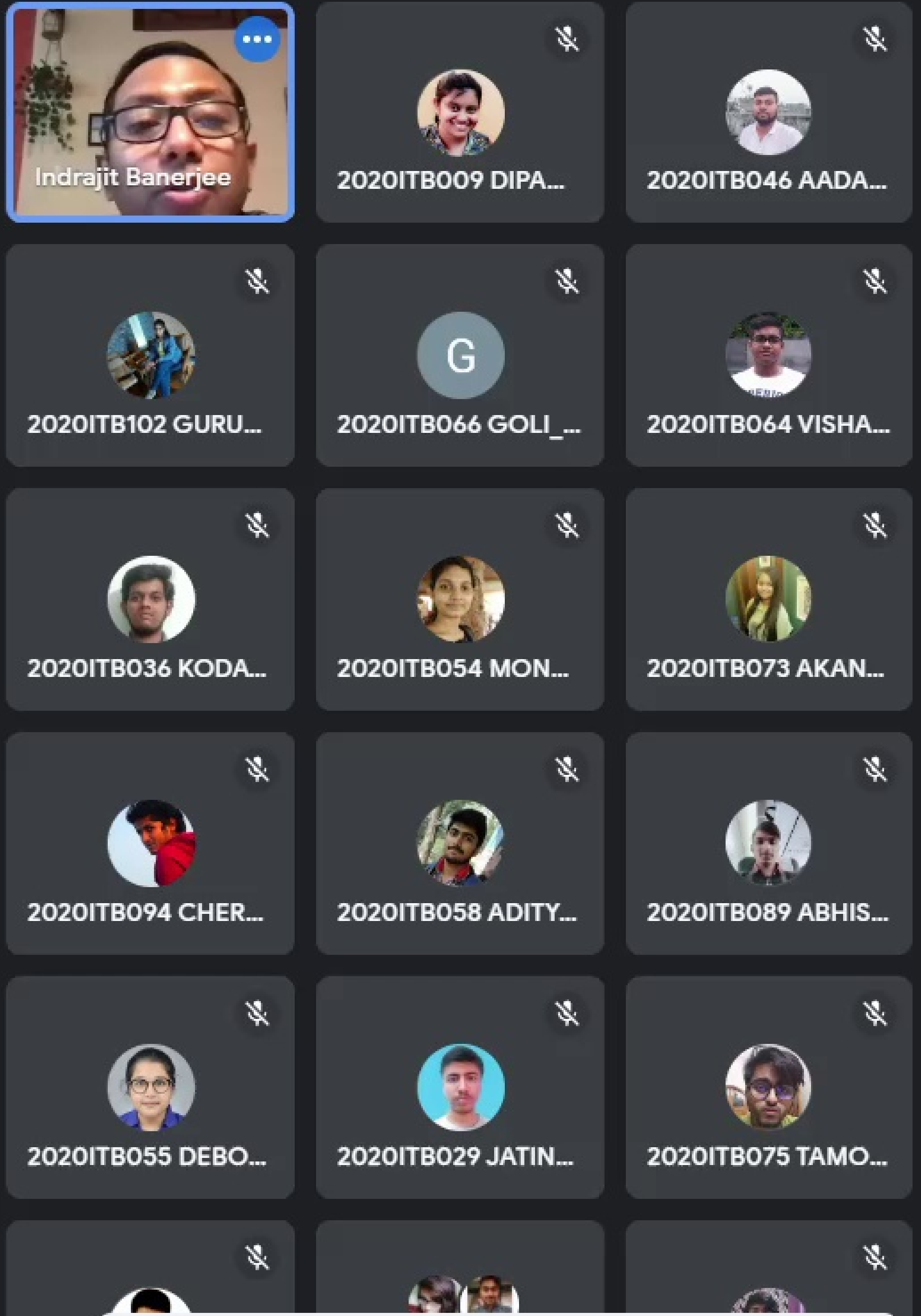
$= (x_P+2)^2 + (y_P - 3/2)^2 + R^2$

$= d_{old} + (2x_P - 2y_P + 5).\ \Delta_{SE} = 2x_P - 2y_P + 5$

- Δ_E and Δ_{SE} Vary at each step unlike the linear case.
- (x_P, y_P) is called the point of evaluation
- Chose the pixel based on the sign of variable d in the previous iteration
- Update the decision variable d with the Δ that corresponds to the choice of pixel.
- Starting point is (0, R).
- The next mid-point is (1, R- ½)
- F (1, R- ½) = 1 +(R² +- R + ¼) - R² = 5/4 – R

```
void MidpointCircle (int radius, int value)
/* Assumes center of circle is at origin */
{
    int x = 0;
    int y = radius;
    double d = 5.0 / 4.0 – radius;
    CirclePoints (x, y, value);

    while (y > x) {
        if (d < 0)              /* Select E */
            d += 2.0 * x + 3.0;
        else {                  /* Select SE */
            d += 2.0 * (x – y) + 5.0;
            y––;
        }
        x++;
        CirclePoints (x, y, value);
    } /* while */
} /* MidpointCircle */
```

Fig. 3.15 The midpoint circle scan-conversion algorithm.



R510819001 JISHU_SHIT has raised a hand
Open queue

10:09 AM | Computer Graphics Theory

60

2 Midpoint circle algorithm

- Bresenham: More efficient that 8-way symmetry. Go all the way round the circle.
- Mid-point algorithm: integer centre and radius.
- Second Octant: Consider 45^0 between x=0 to x=y=R/√2.
- Evaluate mid-point between pixels and find which is closer to the circle.

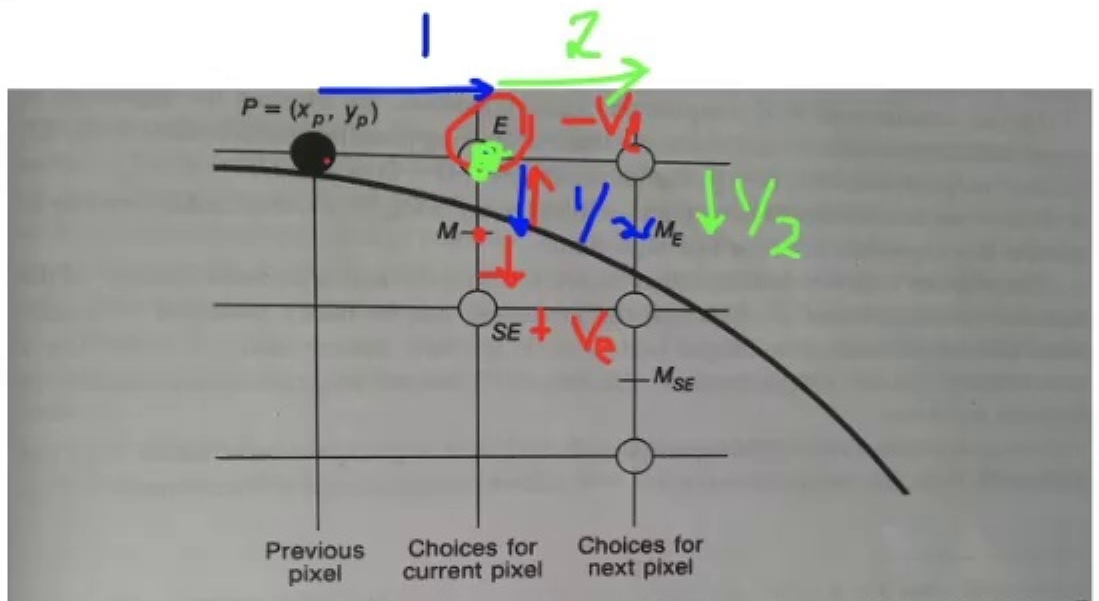


Fig. 3.14 The pixel grid for the midpoint circle algorithm showing *M* and the pixels *E* and *SE* to choose between.

If pixel P is at (x_P, y_P) has been choosen previously select between E and SE.
$F(x,y) = X^2 + y^2 - R^2$ is 0 on the circle +ve outside and –ve inside. If mid-point outside select SE else E