CSCE 441 Computer Graphics

scan conversion of lines

- horizontal, vertical lines are easy
- for general lines, assume $0 < slope < 1$ (flat to diagonal)
  - you can transform any line to fit this
- naive algorithm would just use floating point and round off
  - floating point is sometimes slow (especially back when not every computer did it in hardware)
- slope from two points:
$$m = \frac{y_H - y_L}{x_H - x_L} a$$

- $s\frac{a}{b}a$
- intercept from two points: $b = y_L - m * x_L$
- **Simple Algorithm**
  - start from (xL, yL) and draw to (xH, yH)
    * where $xL < xH$

      ```python
      def draw_line(xL, yL, xH, yH):
          x, y = (xL, yL)
          for i in range(0, xH - xL):
              draw_pixel(x, round(y))
              x = x + 1
              y = m * x + b # simplifies to
              y = y + m
      ```
  - problem: uses floating point math
  - problem: rounding
- **Midpoint Algorithm**
  - given a point, we just need to know whether we will move right or up and right on the next step (N or NE)
  - we can simplify this to whether the actual line travels above or below the point $(x + 1, y + 1/2)$
    * so we derive formula from $y = m * x + b$
  - formula: $f(x, y) = c * x + d * y + e$
    * $c = yL - yH$
    * $d = xL - xH$
    * $e = b * (xL - xH)$
    * $f(x, y) = 0$: (x, y) is on the line
    * $f(x, y) < 0$: (x, y) below line
    * $f(x, y) < 0$: (x, y) above line
  - don't want to recalculate formula at every step, so do it iteratively
    * that is, use $f(x + 1, y + 1/2)$ to calculate $f(x + 2, y + 1/2)$ or $f(x + 2, y + 3/2)$ depending on right or up-right choice last time
  - went right last time, now calculate $f(x + 2, y + 1/2)$

* $f(x+2, y+1/2) = c + f(x+1, y+1/2)$
- went up-right last time, now calculate $f(x+2, y+1/2)$
    * $f(x+2, y+3/2) = c + d + f(x+1, y+1/2)$
- starting value: $f(x+1, y+1/2) = f(xL, yL) + c + (1/2)d = c + (1/2)d$
    * we can eliminate $f(xL, yL)$ because we know it is on the line
    * furthermore, we can use $f(x+1, y+1/2) = 2*c + d$ because multiplying by 2 does not change the sign of f. Also, this saves an expensive division
- full algorithm:
```python
def midpoint_algorithm_line(xL, yL, xH, yH):
    x = xL
    y = yL
    d = xH - xL
    c = yL - yH
    sum = 2*c + d
    draw_pixel(x,y)
    while x < xH:
        if sum < 0:
            sum += 2*d
            y += 1
        x += 1
        sum += 2*c
        draw_pixel(x,y)
```
- pro:
    * only integer operations
    * extends to other kinds of shapes, just need formula to tell if inside/outside shape (called implicit formula)
- same as Bresenham's algorithm (more common algorithm)

scan conversion of polygons

clipping lines

clipping polygons

transformations in 2D

fractals and iterated function systems

transformations in 3D

color

lighting