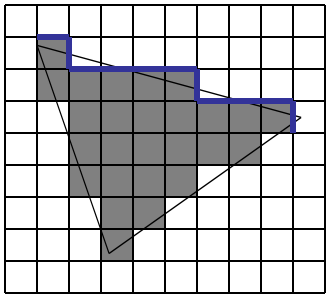
CS 362: Computer Graphics

Anti-Aliasing

Dr. Samit Bhattacharya
Dept. of Comp. Sc. & Engg.
IIT Guwahati, Assam, India

Stair-stepping

- After triangles are scan converted, jagged right angle patterns at the edges can be seen
  - Known as the *stair-stepping* or "*the jaggies*"
- These can be visually distracting, especially for high contrast edges near horizontal or vertical
- Stairstepping is a form of *aliasing*

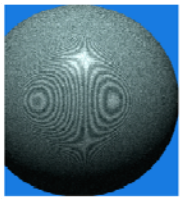


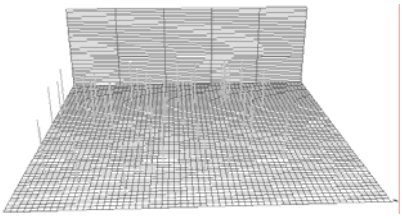
Brightness of Lines

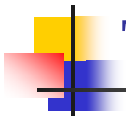
- Lines of different orientation may appear to have different brightness
- Inclined lines appear dimmer than horizontal/vertical lines
- This is due to the difference in pixel distance
  - Distance is higher in slanted lines
- Another example of aliasing

Moiré Patterns

- When we try to render high detail patterns with a lot of regularity (like a grid), we occasionally see strange concentric curve patterns forming
  - Known as Moiré patterns and are another form of *aliasing*





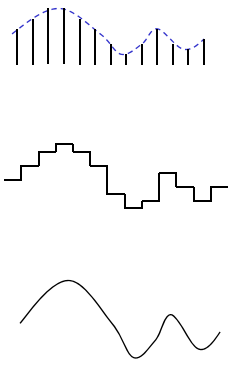


The Propeller Problem

- Consider an animation of a spinning propeller, rendering at 30 frames/second
  - If the propeller is spinning at 1 RPS, then each image shows the propeller rotated an additional 12 degrees, resulting in the appearance of correct motion
  - If the propeller is now spinning at 30 RPS, each image shows the propeller rotated an additional 360 degrees from the previous image, resulting in the appearance of the propeller sitting still!
  - If it is spinning at 29 RPS, it will actually look like it is slowly turning backwards
- Known as *strobing* problem; another form of *aliasing*

CG & Signal Processing

- True intensity treated as continuous signal composed of various frequencies
  - Image as a *continuous signal*
- We need to *sample* it
  - Need some sort of *discreet sampling* technique
- Once we have our sampled signal, we then *reconstruct* it
  - In CG, this reconstruction takes place as a bunch of colored pixels on a monitor



Aliasing

- The examples cover wide range of problems, but they all result from essentially the same thing
  - In each situation, we start with a *continuous signal*
  - We then *sample* the signal at *discreet points*
  - Those samples are then used to *reconstruct* a new signal, that is intended to represent the original signal
  - However, the reconstructed signals are a false representation of the original signals

Aliasing

- In English, when a person uses a false name, that is known as an *alias*, and so it was adapted in *signal analysis* to apply to falsely represented signals
- Aliasing in CG usually results in visually distracting *artifacts*, and a lot of effort goes into trying to stop it. This is known as *anti-aliasing*

Aliasing Types

- Spatial
  - Aliasing problems based on regular sampling in space
  - Example: Jaggies, Moiré pattern
- Temporal
  - Aliasing problems based on regular sampling in time
  - Example: strobing problem

Nyquist Frequency

- Theoretically, in order to adequately reconstruct a signal of frequency *x*, the original signal must be sampled with a frequency of greater than 2*x*
  - This is known as the *Nyquist frequency* or *Nyquist limit*
- However, this is assuming that we are doing a somewhat idealized sampling and reconstruction
  - In practice, it's probably a better idea to sample signals at a minimum of 4*x*

Filtering

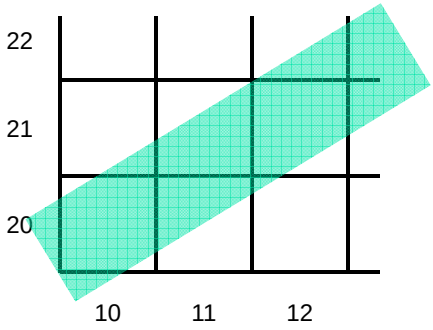
- Uniform regions of constant intensity values correspond to the low frequency components
- Intensity values that change abruptly and correspond to a sharp edge are at the high end of the frequency spectrum
- To lessen jagged contours, we need to smooth out sudden intensity changes
  - Filter out high frequency components

Anti-aliasing

- Technique that compensates for aliasing effects
- Some designed to treat particular artifact
  - Outlined font associated with a set of rules/hints for adjustment/realignment for proper display
    - TrueType
    - OpenType (successor of TrueType)
- General purpose techniques
  - Pre-filtering (area sampling)
  - Post filtering (super-sampling)

Un-weighted Area Sampling

- Set each pixel intensity proportional to the area of overlap of pixel



Foreground and Background

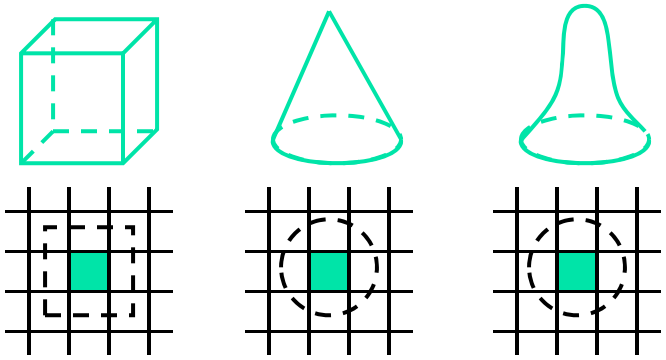
- Compute percent of pixel covered by line, p
- Line color is c_l
- Background color is c_b
- Pixel color is computed as,

$$color = p\ c_l + (1\text{-}p)\ c_b$$

Weighted Area Sampling

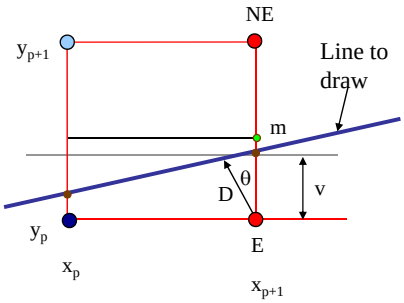
- Treat pixel area as a circle with a radius of one pixel
- Use a radially symmetric weighting function (e.g., cone, bell (Gaussian))
  - Areas closer to the pixel center are weighted more heavily
- Better results than un-weighted, higher cost

Filter Functions (Weighting Surface)



Gupta-Sproull Algorithm

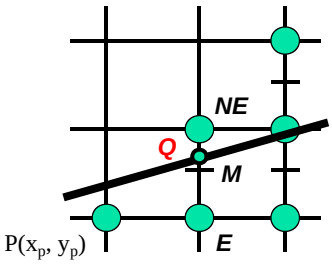
- A pre-filtering technique
- Calculate pixel intensity by computing distance from pixel center to line using the midpoint line algorithm.



Midpoint Line Algorithm

- Decision variable

$$d = F(M)$$
$$= F\left(x_p + 1, y_p + \frac{1}{2}\right)$$
$$= a\left(x_p + 1\right) + b\left(y_p + \frac{1}{2}\right) + c$$



- d > 0 : choose NE

$$d_{new} = F\left(x_p + 2, y_p + \frac{3}{2}\right) : d_{new} = d_{old} + a + b$$

- d <= 0 : choose E

$$d_{new} = F\left(x_p + 2, y_p + \frac{1}{2}\right)$$
$$d_{new} = d_{old} + a$$

Midpoint Line Algorithm

- Initial Value of d

$$F\left(x_0+1, y_0+\frac{1}{2}\right) = a(x_0+1)+b\left(y_0+\frac{1}{2}\right)+c$$

$$= F(x_0, y_0)+a+\frac{1}{2}b$$

$$\rightarrow F(x, y) = 2(ax+by+c)$$

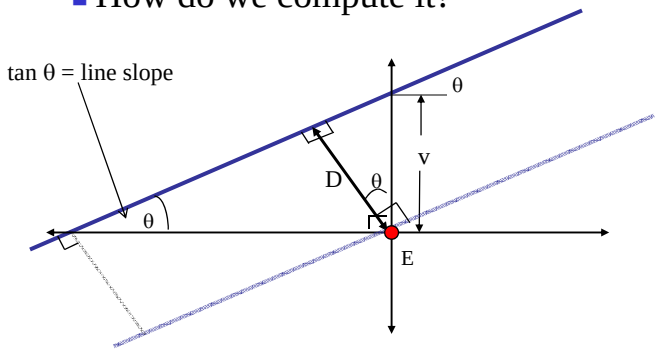
$$\rightarrow d = 2a+b$$

- Update d

$$\text{if } d>0 \text{ then } \begin{cases} x++ \\ y++ \\ d+=2(a+b) \end{cases} \qquad \text{if } d<=0, \text{ then } \begin{cases} x++, \\ d+=2a \end{cases}$$

Gupta-Sproull Algorithm

- Let E is selected by the mid point algorithm
  - D is the perpendicular distance from E to the line
- How do we compute it?



$\Delta x = x_{end} - x_{start}$

$\Delta y = y_{end} - y_{start}$

$$\cos\theta = \frac{\Delta x}{\sqrt{\Delta x^2+\Delta y^2}}$$

$$\cos\theta = \frac{D}{v}$$

$$D = \frac{v\Delta x}{\sqrt{\Delta x^2+\Delta y^2}}$$

Gupta-Sproull Algorithm (cont)

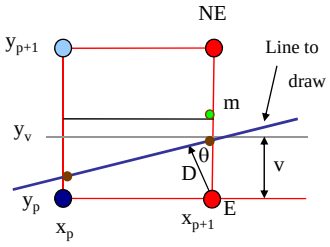
Recall from the midpoint algorithm:

$$f(x,y) = 2(ax + by + c) = 0$$

So, $y = -\frac{ax+c}{b}$

and $y_v = -\frac{a(x_p + 1) + c}{b}$

$v = y_v - y_p$



Therefore $v = -\frac{a(x_p + 1) + c}{b} - y_p$

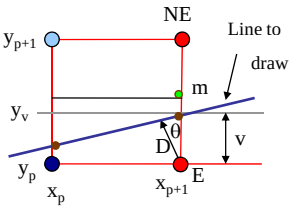
Gupta-Sproull Algorithm (cont)

So $-bv = a(x_p + 1) + c + by_p$

From the midpoint computation,

$b = -\Delta x$

So:



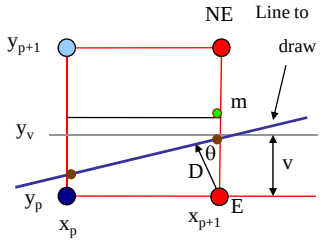
$$v\Delta x = a(x_p + 1) + by_p + c = \tfrac{1}{2} f(x_p + 1, y_p)$$

From the midpoint algorithm, we have the decision variable

$$d = f(m) = f(x_{p+1}, y_p + \tfrac{1}{2})$$

Gupta-Sproull Algorithm (cont)

Going back to our previous equation:

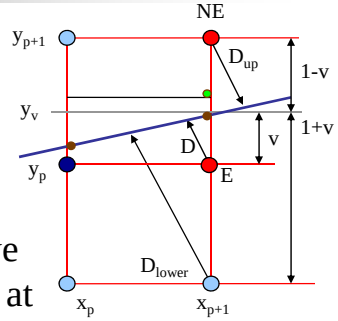


$$
\begin{aligned}
2v\Delta x &= f(x_p + 1, y_p) \\
&= 2a(x_p + 1) + 2by_p + 2c \\
&= 2a(x_p + 1) + 2b(y_p + \tfrac{1}{2}) - 2b\tfrac{1}{2} + 2c \\
&= f(x_p + 1, y_p + \tfrac{1}{2}) - b \\
&= f(m) - b \\
&= d - b \\
&= d + \Delta x
\end{aligned}
$$

Gupta-Sproull Algorithm (cont)

So,

$$D = \frac{v\Delta x}{\sqrt{\Delta x^2 + \Delta y^2}} = \frac{d + \Delta x}{2\sqrt{\Delta x^2 + \Delta y^2}}$$



The denominator is constant.

Since we are blurring the line, we also need to compute the color at the pixels above and below the E pixel

$$D_{up} = \frac{(1 - v)\Delta x}{\sqrt{\Delta x^2 + \Delta y^2}} \qquad D_{lower} = \frac{(1 + v)\Delta x}{\sqrt{\Delta x^2 + \Delta y^2}}$$

Gupta-Sproull Algorithm (cont)

If the NE pixel had been chosen:

$$
\begin{aligned}
2v\Delta x &= f(x_p+1, y_p+1) \\
&= 2a(x_p+1)+2b(y_p+1)+2c \\
&= 2a(x_p+1)+2b(y_p+\tfrac{1}{2})+2b\tfrac{1}{2}+2c \\
&= f(x_p+1, y_p+1/2)+b \\
&= f(m)+b \\
&= d+b \\
&= d-\Delta x
\end{aligned}
$$

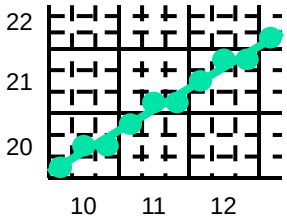
$$D = \frac{v\Delta x}{\sqrt{\Delta x^2+\Delta y^2}} = \frac{d-\Delta x}{2\sqrt{\Delta x^2+\Delta y^2}} \qquad D_{up} = \frac{(1-v)\Delta x}{\sqrt{\Delta x^2+\Delta y^2}} \qquad D_{lower} = \frac{(1+v)\Delta x}{\sqrt{\Delta x^2+\Delta y^2}}$$

Gupta-Sproull Algorithm Summary

- Compute midpoint line algorithm, with the following alterations at each iteration:
- At each iteration of the algorithm:
  - If the E pixel is chosen $D = \frac{d+\Delta x}{2\sqrt{\Delta x^2+\Delta y^2}}$
  - If the NE pixel is chosen $D = \frac{d-\Delta x}{2\sqrt{\Delta x^2+\Delta y^2}}$
  - Update d as in the regular algorithm
  - Color the current pixel according to D
  - Compute $D_{up} = \frac{(1-v)\Delta x}{\sqrt{\Delta x^2+\Delta y^2}}$ $D_{lower} = \frac{(1+v)\Delta x}{\sqrt{\Delta x^2+\Delta y^2}}$
  - Color upper and lower pixels accordingly

Super-sampling

- Divide pixel up into “sub-pixels”: 2×2, 3×3 etc.
  - Increase resolution
- Count no of sub-pixels through which the line passes
  - Set pixel intensity proportional to this count



(10, 20): Maximum Intensity

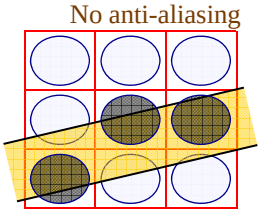
(11, 21): Next Highest Intensity

(11, 20): Lowest Intensity

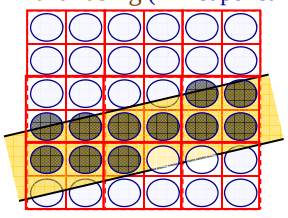
Super-sampling

- We can also consider lines having finite width – (usually 1-pixel wide)
- Sub-pixel is colored if inside line
  - Lower left corner of sub-pixel inside
- Pixel color = average of its sub-pixel colors

No anti-aliasing



Anti-aliasing (2×2 super-sampling)



Pixel-Weighting Masks

- Give more weight to sub-pixels near the center of a pixel area
  - Distribute weights

1	2	1
2	4	2
1	2	1

Example:
• Central sub-pixel has 4 times more weight that corner ones.
• While calculating pixel intensity, the central sub-pixel intensity contribution $1/4^{th}$, $1/8^{th}$ each for the top, bottom, left and right sub-pixel intensities and $1/16^{th}$ each for the rest (corner sub-pixels)
• It's possible to include contribution from neighboring sub-pixels (extend the grid)