1996-11-21 Bo Nordin (bosse&cb.uu.se)

Lecture notes, computer graphics (F11) (Foley 3.12,3.14)

Text (3.12)

Defining and clipping characters
Either define each character a a *curved or polygon outline* and scan convert as needed
or specify each character as a small rectangular *bitmap*.

Then use pixelcopy to generate a character (font cache in invisible screen memory)
When displaying characters all pixels are copied and clipped pixel by pixel.
Or you can clip each characters or even string on the bounding box (this is probably a good opening strategy in all cases).

In SRGP's font cache, each character is stored side by side in a long rectangular strip:

```
typedef struct { int leftX, width; } charLocation;

typedef struct {
int cache;
int descenderHeight;  /* lines used only by descenders */
int totalHeight;          /* the height of each character 7, 9, 12,...*/
int interCharacterSpacing;
charLocation loc[128]]
} fontCache;
```

Simpler but uglier with fixed width
Proportional otherwise
If space must be reduced g, j, etc. may be stored with an extra bit for shift downwards (1/3)
If multi-level device store different intensities (anti-aliasing).

Here the height of all characters is constant while the width varies
A single font with 12 sizes and four faces (Roman, italics,bold, bold italic) takes 48 caches.
Bold made simple: write twice shifted in x
Italics made simple: shift the lines

One can store characters in a device-independent form using polygons, curves (splines)
Use transformations (scaling, shearing)

Note, however, not all point sizes may be obtained from scaling, since the shape of a good-looking character depends on the point size.
One can define each character as a polygon which may be filled
Also, scan conversion of splined text requires far more processing.

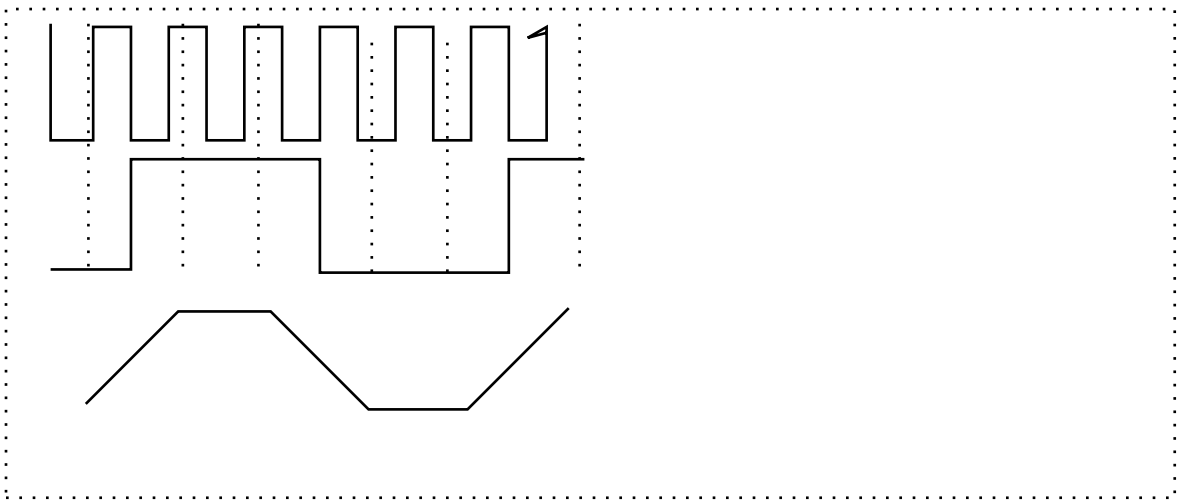
A compromise is to store the fonts in outline form and convert the ones being used in a given application to bitmaps

Anti-aliasing intro (3.14.1 plust material not in the textbook)

What is aliasing??
Aliasing effects exist because the because lines, polygon edges, colour boundaries, etc. are continuous whereas a raster device in discrete.

jaggies staircasing

When displaying a line, etc. on a raster device it must be *sampled* at discrete locations:



In the figure two complete different signals give rise to the same figure after sampling.
The lowermost figure is called an *alias* of the upper, hence aliasing.
In this case the high-frequency signal has been undersampled.
Sampling theorem: In order to prevent aliasing, a signal must be sampled at a rate at least twice the highest frequency in the signal. Undersampling causes highly periodic images to be rendered incorrectly (Nyquist sampling frequency).

Consequences of aliasing:
jagged edges, incorrectly rendering of fine detail or texture, elimination of small objects.
strange quirks in animation (small objects to the left, animation to the right)

Fundamentally there are three methods of antialiasing:
1/ increase the sample rate - i.e. the resolution of the raster (but there is a limit)
2/ treat a pixel as a finite area rather than a point (equivalent to prefiltering the image)
prefiltering (area sampling) no subpixel intensities
3/ postfiltering (supersampling)

Pixel averaging (this is really image analysis!)
each pixel is divided into subpixels to form a higher resolution raster

Increasing resolution
pg 120: as jags increase in number they decrease in size

Unweighted area sampling (3.14.2)
the ideal primitive has zero width but what we draw has width at least == 1 (if vertical or horizontal). otherwise width varies over the primitive

Clearly a line a line should not set the intensity of only a single pixel in a column but

should contribute some amount of intensity to each pixel it intersects (we need a multi-level device for this). Only hor/ver lines would affect exactly 1 pixel per column/row.

Assume that pixels form an array of nonoverlapping square tiles covering the screen centred on grid points. Also assume that a line contribute to the intensity in proportion to the proportion of the pixel's tile it covers. (page 121)

a) The intensity of a pixel intersected by a line edge decreases as the distance increases
b) A primitive cannot influence the intensity at a pixel if it doesn't intersect the pixel.
c) Areas contribute equally regardless of where it is: a small area in the corner of the pixel contributes just as much as does an equal-sized area near the pixel's centre.

Anti-aliasing always makes edges fuzzier.

Weighted area sampling (3.14.3)

We keep a) and b) but not c)
A small area closer to the pixel centre has greater influence.
Now, The pixel represents a circular area larger than the square tile; the primitive will intersect this larger area and thus contribute to the intensity of the pixel.
Areas associated with different pixels actually overlap

weighting function that determines the influence of a pixel of a given small area of a primitive as a function of dA's distance from the centre of the pixel.
Constant for unweighted area sampling and decreases with distance for weighted.
W(x,y) whose height above the (x,y) plane gives the weight.
in weighted area sampling the intensity contributed by each small area is proportional to the area multiplied by the weight.

filter function (*box filter* for unweighted area sampling)
Max intensity is I_{max} which means that for a box with height 1 the total intensity is $I_{max} * 1 = I_{max}$.

A weighting function for weighted sampling:: a function with a maximum at the centre and linear decrease. Circular cone (the base is called *support*).
Should have a larger base than expected: radius 1. Supports overlap which means that small primitives may influence several pixels.
The volume under the entire cone is normalized to 1. Net effect is to decrease the contrast between adjacent pixels and give smoother transitions.
In particular: a horizontal and vertical line of unit thickness is smeared over several pixels.
IMPORTANT: rotational symmetry and linear decrease
rotational symmetry makes is independent of the angle of the line
conic filters are *not* optimal. (optimal most expensive, box least, cone compr).

Draw weighting functions from Hearn/Baker page 175.

Simple example (not in the textbook, but compare Example 3.1, page 76)

We saw before in the 2-pixel variant a simple example of anti-aliasing

Use different levels
We have two candidate points: E and NE
Set the level according to how close to the line they are: $I_{max} = I_E + I_{NE}$
Problem if $I_E \sim I_{NE} \sim I_{max} / 2$ Both pixels get the same half-bright intensity.
Intensity in the unselected pixel: I_{pix} and in the other $I_{max}-I_{pix}$.

Apply a correction factor to I_{pix}:

$1 + k (1 - abs(1 - 2I_{pix})/I_{max}))$

$I_{pix} \sim I_{max}/2 \Rightarrow 1 + k \quad I_{pix}$ small $\Rightarrow 1$

The Gupta-Sproull algorithm (AA)

Precomputes the subvolume of a normalized filter function defined by lines at various distances from the pixel centre and store them in a table.
Table lookups are often used to reduce computation time.
Use circular pixel area with radius == 1 (a grid unit)
A line of unit thickness with slope < 1 intersects 3 supports in a column
Table lookup for Filter(D,t) (t depends on the thickness, we assume 1)
Filter depends on the filter function (we assume conical)
0 outside the support, for D > 1.5.
Modified scan conversion:

Using similar rectangles and knowing that the slope is dy/dx:
D is computed as v cosϕ = vdx / sqrt($dx^2 + dy^2$)
The vertical distance is the difference between y coordinates (signed!!)\
We should pass the absolute value to the Filter function
The selected pixel is always the middle pixel.
The pixel above is 1-v from the line, the one below is 1+v from the line.
This is true regardless of the relative position.

We don't want to compute v directly, we use $d = F(M) = F(x_P+1,y_P+1/2)$.
If we know the x coordinate of a point on the line we can compute y from:

$F(x,y) = 2(ax+by+c)=0 \quad y = (ax+c)/-b$

For pixel E: $x = x_P+1$ and $y = y$, and $v = y - y_P$
$v = ((a(x_P+1)+c)/-b) - y_P$
$-bv = a(x_P+1) + by_P + c = F(x_P+1,y_P)/2$
$b = -dx$: $vdx = F(x_P+1,y_P)/2$

vdx is the numerator of D = ... above, Its denominator can be precomputed
we want to compute vdx incrementally and avoid division:

$2vdx = F(x_P+1,y_P) = 2a(x_P+1) + 2by_P + 2c =$ (express in d, add 1/2, subtract 1/2)
$2a(x_P+1) + 2b(y_P+1/2) - 2b/2 + 2c = d + dx$

Thus, D = (d + dx) / (2 * sqrt(dx^2+dy^2)) and the denominator is 1 / (2*sqrt(dx^2+dy^2))
Numerators for the others are then:

$2(1-v)dx = 2dx - 2vdx$ and $2(1+v)dx = 2dx + 2vdx$

for NE: $2vdx = F(x_P+1, y_P+1) = d - dx$ and numerators for the others are the same

Polygons: keep as they are and do post-processing

Supersampling

alias effects decrease resolution: i.e. use higher resolution for the algorithm that generates the picture (1280*1024 -> 5120*4096)
When mapping to the screen average over the higher-resolution pixel.
Ex: 4 bit planes. If high-resolution image is binary n = 16

pixel = round ((n^2-1) # of pixels)/n^2) (n^2-1 because we want 0,15)

DRAW ex (area) from Burger/Gillies:

DRAW ex (line) from Hearn/Baker pg 173

Works well for filled areas, less well for lines (too low)
Supersampling is computationally heavy

Filtering
Also called convolution (this is really image analysis)
The value of a pixel in the anti-aliased image is computed as the weighted sum of its neighbours with the weight inversely related to distance.
1/36 1/9 1/36
1/9 4/9 1/9
1/36 1/9 1/36

$$V(i,j) = \sum_{x,y=-1}^{x,y=1} \Sigma I(x,y) * f(x,y)$$

More weight in them middle: more like the original
less weight in the middle: more smoothing
Larger filter - increased effect (100x100 -> blur)
Effective implementation (it is possible to make the speed independent of filter size)

The filter is moved over the image
Border problems
It is possible to filter a supersampled image
Non-linear filters: median (preserves edges, but is slow...)

You can see examples of this when/if you visit CBA

Scan conversion summary:
speed is essential: incremental algorithms integer-only in inner loops
The basic algorithms can be extended to handle thickness as well as patterns and fill areas

Single-pixel primitives minimize the error, thick primitives can trade of quality/correctness for speed

Much of 2D raster graphics today operate on single-bit-per-pixel primitives (even on colour displays) real-time antialiasing will soon become prevalent