Direct Rendering

May 2, 2005

Direct Rendering Goals

- General Goals
  - Small memory footprint
  - Fast rendering
  - High-quality results identical to those of Saffron – V1 using distance-based anti-aliasing and alignment zones

Direct Rendering Goals

- Specific Goals
  - Avoid explicit ADF generation
  - Compute a minimum number of distances
    - Only compute distances near the edge of a glyph (i.e., within the "filter area")
    - Set areas outside the filter area without computing distances
    - For each sample point within the filter area, only compute the distance to the nearest "feature" (line, curve, corner)
    - Avoid computing distances to the endpoints of lines and curves
    - Eliminate bounding box tests

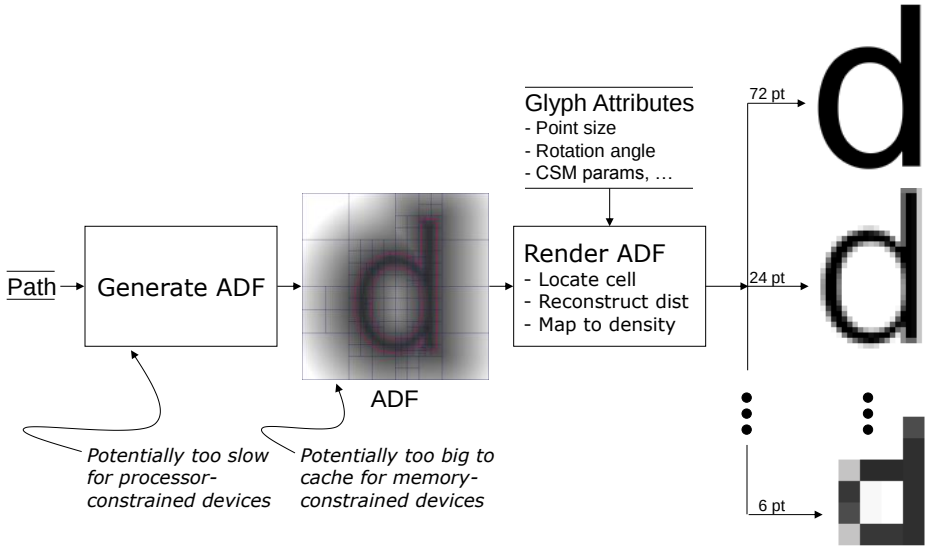
Direct Rendering Goals

- Specific Goals
  - Amenable to CPU, GPU, and ASIC implementations
  - Suitable for outlines and stroke-based formats
  - Solve known issues
    - Robust predicate test for special cells
    - Support gross deformation and non-uniform scaling

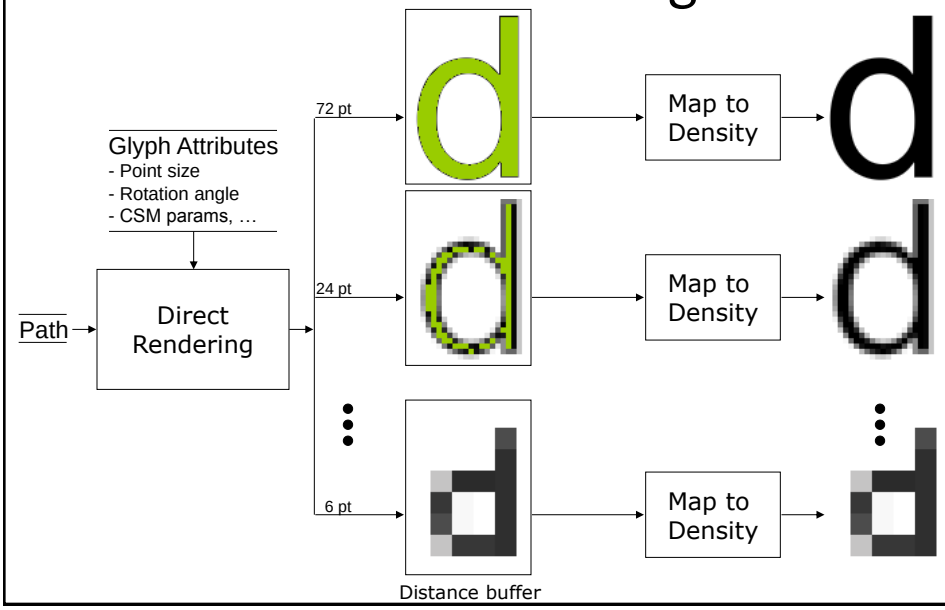
Direct Rendering

- Direct rendering vs. explicit ADF generation
  - An explicit ADF representation computes distances for ***all*** potential density images and thus can be used to render multiple density images of the same glyph
    - Different point sizes
    - Different rotation angles
    - Different CSM parameters
      - Varying stroke weight and edge sharpness
  - Direct rendering ***only*** computes distances needed for a particular density image
    - Produces the density image for a single set of glyph attributes **as efficiently as possible**

Explicit ADF Generation



Direct Rendering

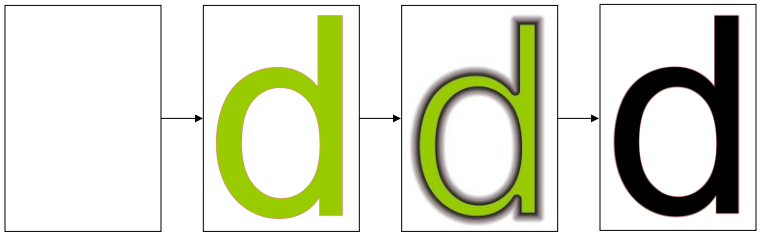


Direct Rendering

- Basic approach
  - Divide the distance field for each glyph into three areas
    - Outside area
    - Inside area
    - Filter area
  - Render each area separately to determine distances, combining these distances in a distance buffer the size of the density image
    - Initialize the buffer to a large negative value (< outside cutoff)
    - Fill the inside area with a large positive value (> inside cutoff)
    - Compute distances in the filter area and combine in the buffer
  - Map distances in the distance buffer to density values

Direct Rendering

- Basic approach



Initialize distance buffer to large neg. value (< outside cutoff)

Fill inside with large pos. value (> inside cutoff)

Compute distances in the filter area and combine in the distance buffer

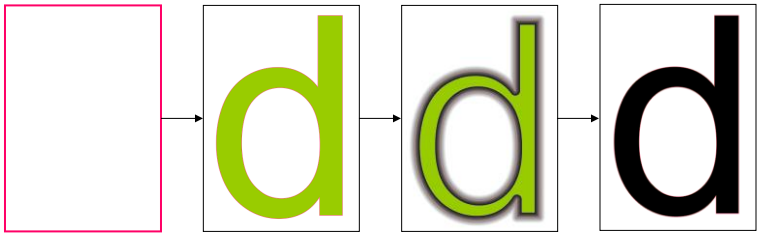
Map distances to density values

Direct Rendering

- Basic approach
  - The geometry defining the filter area
    - Can be very coarse (a fine sweep is unnecessary)
    - Is fast to determine
    - Is composed of simple shapes which can be decomposed into primitive shapes (e.g., triangles and quadrilaterals) which are fast and simple to render on CPUs, GPUs, and ASICs
  - Exploit the mathematical properties of distance fields and features (lines, corners, curves) to compute distances within the filter area very efficiently
  - For each sample point within the filter area, only computes distances to features within the filter radius

Direct Rendering

- Step 1: Setting the outside area



Initialize distance buffer to large neg. value (< outside cutoff)

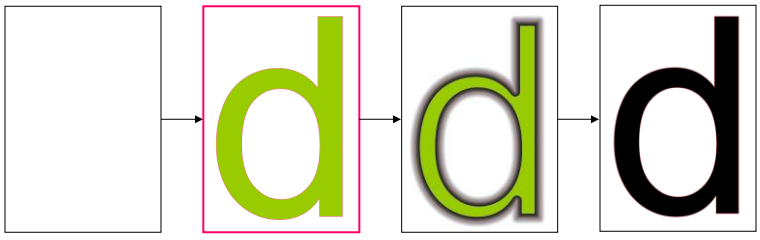
Fill inside with large pos. value (> inside cutoff)

Compute distances in the filter area and combine in the distance buffer

Map distances to density values

Direct Rendering

- Step 2: Filling the inside area



Initialize distance buffer to large neg. value (< outside cutoff)

Fill inside with large pos. value (> inside cutoff)

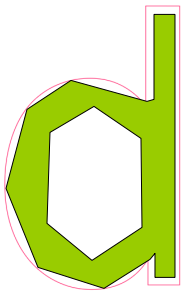
Compute distances in the filter area and combine in the distance buffer

Map distances to density values

Direct Rendering

- Filling the inside area
  - Two approaches
    - Rasterize the original outline using a binary fill
    - Rasterize a polygonal shape approximating the inside area

Rasterize the outline using a binary fill



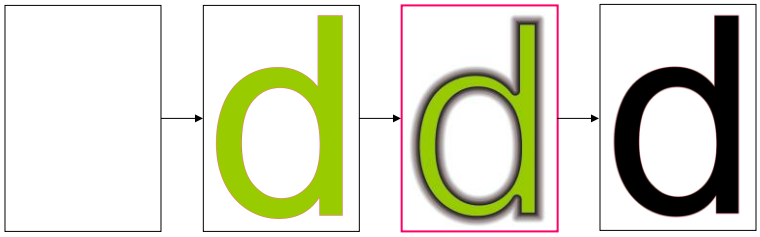
Approximate the inside area with a simpler shape

Direct Rendering

- Filling the inside area
  - The filled inside area can overlap the filter area
    - Distance values computed from overlapping areas are combined in the distance buffer
      - Combining chooses the smallest distance value
    - Distances in the filter area will be smaller than the inside fill value so they will be chosen over the inside value during combining

Direct Rendering

- Step 3: Computing distances in the filter area



Initialize distance buffer to large neg. value (< outside cutoff)

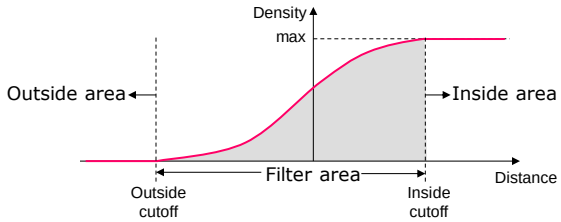
Fill inside with large pos. value (> inside cutoff)

Compute distances in the filter area and combine in the distance buffer

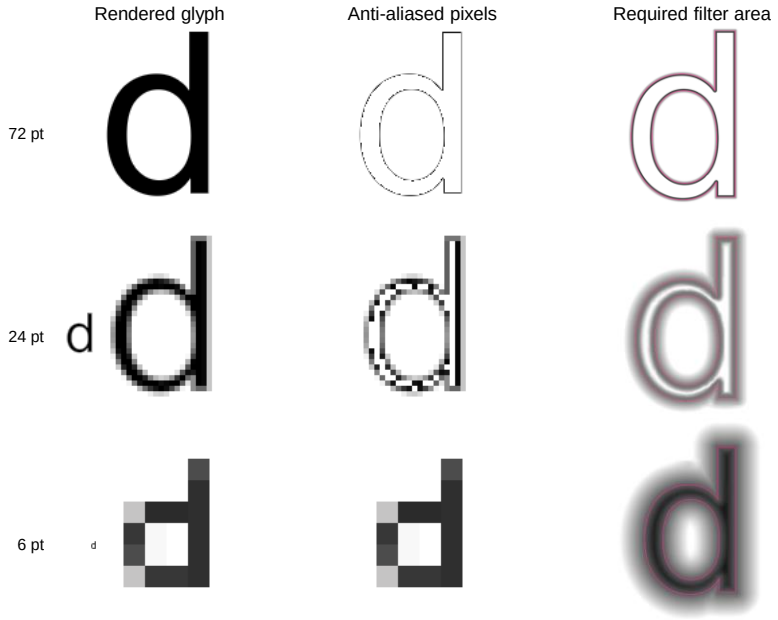
Map distances to density values

Direct Rendering

- Filter area
  - The region of the distance field that is mapped to a density value between zero and one
    - Depends on the CSM parameters (inside and outside cutoff) and point size



  - This approach produces the same density image as Saffron – V1 (i.e., explicit ADF generation)
    - **Identical density image yields identical type quality**



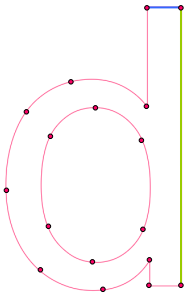
The filter area depends on point size

Direct Rendering

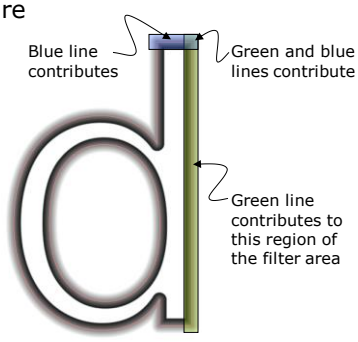
- Computing distances in the filter area
  - Consider each feature (line, corner, curve) independently
    - Define a region bounding the feature's contribution to the filter area
    - Compute the distance to the feature for each density image pixel (or pixel component) in that region
  - Combine distances in the distance buffer
    - Choose the smallest distance for each pixel (or pixel component)
    - This provides accurate distance values → **no quality compromise**

Direct Rendering

- Efficient distance computation
  - Each feature contributes to a limited region of the filter area
  - The distance near more than one feature is the smallest of the distances to each nearby feature

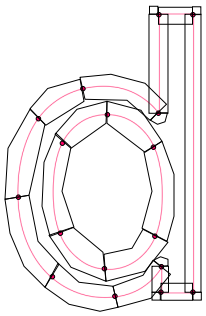


Glyph outline



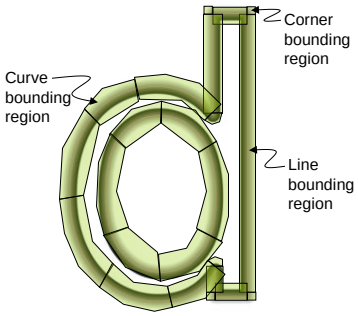
Filter area

Direct Rendering



Define regions bounding each feature's contribution to the filter area

- Regions can overlap
- Regions can be too big (but not too small)
- Choose simple shapes that are easy to determine and to rasterize



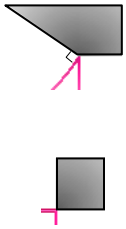
Compute distances for each region and combine them in the distance buffer

- In each region, compute distance to a single feature
- Combining chooses the smallest distance value for each pixel or pixel component

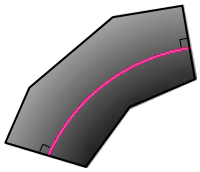
Direct Rendering

- Regions for bounding features

Line bounding region
- Perpendicular to line at endpoints
- Spans outside to inside cutoffs
- Distance field is linear



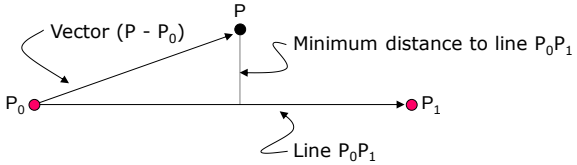
Corner bounding regions
- Limited by normals of adjacent edges
- Diameter determined by filter cutoffs
- Distance field is quadratic



Curve bounding region
- Perpendicular to curve at endpoints
- Polygonal shape
- Based on convex hull of quadratic curve (easy to determine)

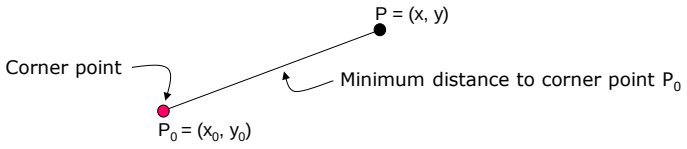
Direct Rendering

- Computing distances in feature regions
  - Lines
    - Analytic method
      - Rasterize quadrilateral region bounding the line
      - Compute the distance at each sample point P inside the quadrilateral region according to dist(P) = $(P - P_0) \times (P_1 - P_0) / ||(P_1 - P_0)||$
      - Note that the quadrilateral region is constructed to avoid
        - » Endpoint distance computations
        - » Conditionals



Direct Rendering

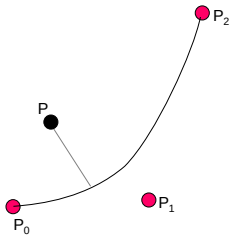
- Computing distances in feature regions
  - Corners
    - Analytic method (use squared distances to avoid square roots)
      - Rasterize triangles partitioning the region bounding the corner
      - Compute the distance at each sample point P inside each triangle according to squaredDist(P) = $||(P - P_0)||^2 = (x - x_0)^2 + (y - y_0)^2$



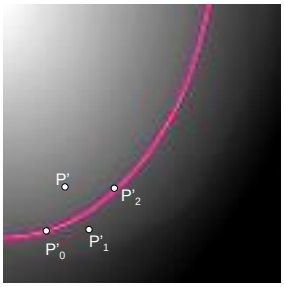
Direct Rendering

- Computing distances in feature regions
  - Curves
    - Texture mapping (use squared distances to avoid square roots)
      - Rasterize triangles partitioning the region bounding the curve
      - Compute the distance at each sample point P inside each triangle by
        - » Mapping P to P' in a canonical parabola (i.e., $y = x^2$) distance field stored in a texture map
        - » Texture map contains, for every point in space, the vector (dx,dy) to the closest point on the canonical parabola
        - » Determining (dx', dy') at P' from the texture map using bilinear interpolation
        - » Mapping (dx', dy') to (dx, dy) in the coordinates of the sample point
        - » Computing squaredDist(P) = $dx^2 + dy^2$

Direct Rendering



Curve with control points P_0, P_1, and P_2. Sample point P.



Canonical parabola distance field and transformed sample point P' and curve control points

Direct Rendering

- Computing distances in feature regions
  - Curves
    - Analytic method (use squared distances to avoid square roots)
      - Rasterize triangles partitioning the region bounding the curve
      - Compute the distance at each sample point P inside each triangle using a cubic root finder
      - Cubic root finder can be accelerated by mapping the sample point P to a canonical parabola ($y = x^2$) which defines convergence regions and an initial guess for the solution

Direct Rendering

- Back-of-the-envelope calculation
  - 200 MHz ARM 9 processor
    - 66K Gouraud-shaded texture-mapped triangles per second when rendering a full-screen (320x320) 3D model
  - Direct rendering rates
    - Average outline: 10 corners, 10 lines, 15 Bezier curves
    - 3 triangles per corner, 2 per line, 5 per curve
      ➔ 125 triangles to represent the filter area
    - Can direct render 66K/125 = 520 **full screen** glyphs (320x320) per second
    - Glyphs occupying less screen space (typical point sizes) would render **significantly** faster
    - With a density image cache that delivers a 90% hit rate, rendering throughput increases by 10x

Direct Rendering Summary

- Accurate
  - Given a set of glyph attributes, direct rendering produces the same density image as would be produced via explicit ADF generation
- Fast
  - Minimum number of distance computations
    - Only compute distances in the filter area
    - Filter area is fast to determine and can be decomposed into primitive shapes that are fast and simple to render on CPUs, GPUs, and ASICs
    - Computing distances within each primitive shape is fast and simple
- Low memory overhead
  - Compute distances directly from outlines or strokes
    - Process each feature independently
    - Buffer for combining distances the same size as the final density image