```
//-----------------------------------------------------------------------------------
//  Filename: ADFImplicitFloat.c
//-----------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------
//  Copyright 2004-2008 Mitsubishi Electric Research Laboratories (MERL)
//  An implementation for processing (e.g., generating and rendering) implicit ADFs
//  Ronald Perry, Sarah Frisken, and Eric Chan
//-----------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------
//  This file contains the floating point implementation for processing implicit
//  ADFs. The corresponding fixed point implementation for processing implicit ADFs
//  is contained in ADFImplicitFixed.c, which is modelled after the floating point
//  implementation. Consequently, any changes made to this file must be reflected
//  appropiately in ADFImplicitFixed.c.
//-----------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------
//  Throughout this file, suggestions for possible implementation changes are
//  annotated as development notes, i.e., DevNotes.
//-----------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------
//  Overview
//
//      This file contains the following major functional blocks:
//
//          (1) Implicit ADF generation
//          (2) Implicit ADF rendering
//          (3) Support for implicit ADF SAZ alignment zone detection
//          (4) Implicit ADF validation
//
//      Implicit ADF generation converts a given ADFPath from font units to ADF
//      coordinates. The resultant representation (viewed as an opaque ADF to the
//      application but considered a preprocessed ADFPath internally) can effectively
//      be scaled, translated, rotated, sheared, etc. (by setting the appropriate
//      ADFRenderSetup() attributes) prior to rendering without affecting the quality
//      of the rendered image. Note also that the preprocessed ADFPath (i.e., the
//      generated ADF) can be cached as an ADF using the library's dual caching
//      system.
//
//      Implicit ADF rendering generates an implicit ADF from a given preprocessed
//      ADFPath (viewed as an opaque ADF to the application), renders the implicit
//      ADF into a distance buffer using the specified rendering attributes, and then
//      maps distances in the distance buffer to density values. These density values
//      are packed into pixels of a specified density image. Implicit ADF rendering
//      of an outline-based glyph comprises the following steps:
//
//          (1) Each pixel (for CRT rendering) or each pixel component (for LCD
//              rendering) of the specified density image is cleared to zero.
//
//          (2) A distance buffer is created large enough to store a distance sample
//              for each pixel or pixel component of the specified density image. A
//              distance buffer is a 2D array of floating point values that is used
//              to combine distances from contributing elements (i.e., line segments
//              and corners) of the preprocessed ADFPath at each sample point. Each
//              distance sample in the distance buffer is initialized to
//              LARGE_OUTSIDE_DIST_VAL. Note that a distance value of
//              LARGE_OUTSIDE_DIST_VAL maps to zero density in the density image.
//
//          (3) Pen commands in the preprocessed ADFPath are transformed from ADF
//              coordinates to floating point image coordinates. If requested, MAZ
//              alignment zone detection and grid fitting are performed on the
//              transformed pen commands (see ADFAlgnZonesMAZ.h,
//              ADFAlgnZonesMAZOutlines.c, and ADFAlgnZonesMAZStrokes.c for details).
//
//          (4) An internal path is created from the transformed pen commands. The
//              internal path is used to generate and render implicit ADF boundary
//              cells and to rasterize the glyph interior. Curvto commands in the
//              transformed pen commands are not added to the internal path, but
```

```
//              instead are replaced by a sequence of lineto commands that closely
//              approximate the corresponding curve segment. Subdividing curve
//              segments into line segments has significant advantages in performance
//              and ease of implementation over processing curve segments directly
//              (see CreateInternPath() for more details).
//
//          (5) Each element of the internal path is processed. For each element, an
//              implicit ADF boundary cell is generated. The implicit ADF boundary
//              cell includes a representation of the cell's geometry (e.g., its
//              vertices) and data required to compute the minimum unsigned Euclidean
//              distance from any point inside the implicit ADF boundary cell to the
//              element represented by the implicit ADF boundary cell. Implicit ADF
//              boundary cells are represented in floating point image coordinates.
//              The size and geometry of an implicit ADF boundary cell is constructed
//              such that it covers the area obtained by sweeping a line segment
//              along the section of the internal path corresponding to the element
//              represented by the implicit ADF boundary cell, where the swept line
//              segment is perpendicular to the element and extends on each side of
//              the element by the specified CSM filter cutoff values. Consequently,
//              the union of the geometry of the implicit ADF boundary cells for all
//              elements of the internal path is guaranteed to cover those sample
//              points which require antialiasing. By computing distances only near
//              the edge of a glyph, the time required to render the glyph is
//              minimized. As each implicit ADF boundary cell is generated, the
//              implicit ADF boundary cell is rendered into the distance buffer by
//              rasterizing the cell interior, determining the minimum unsigned
//              Euclidean distance from each sample point to the element represented
//              by the implicit ADF boundary cell, and combining the determined
//              distance value with the corresponding distance sample stored in the
//              distance buffer. The combining selects the minimum magnitude
//              distance. To reduce the number of comparisons, all determined
//              unsigned distances are converted to negative values and distances are
//              combined by choosing the maximum negative distance. Distances
//              corresponding to sample points inside the ADF glyph are converted to
//              positive values in the next step (i.e., step (6) of rendering).
//
//          (6) The interior of the implicit ADF glyph is rasterized and distances in
//              the distance buffer for sample points inside the glyph are converted
//              to positive values, thereby completing the computation of an
//              adaptively sampled signed distance field of the implicit ADF glyph.
//
//          (7) Distances in the distance buffer are mapped to density values by
//              applying the given CSM parameters. Color reduction is applied to
//              these density values during LCD rendering if enabled. The final
//              density values are packed into the specified ADFImage, where the
//              packing depends on the display mode.
//
//      Implicit ADF rendering of a uniform-width stroke-based glyph differs from
//      implicit ADF rendering of an outline-based glyph in the following ways:
//
//          (a) Before the elements of the internal path are processed as described
//              above in step (5), the CSM inside and outside filter cutoff values
//              are both decreased by half the stroke width. These adjustments are
//              required to account for the stroke width of the uniform-width
//              stroke-based glyph. Effectively, this step moves the filter position
//              outward (i.e., away from the centerlines of the glyph) by half the
//              stroke width without changing the CSM filter width.
//
//          (b) Step (6) (i.e., the rasterization of the interior of the implicit ADF
//              glyph) is skipped. In the case of a uniform-width stroke-based
//              glyph, the interior is rendered during step (5) (i.e., during the
//              processing of each element of the internal path). Therefore, it is
//              unnecessary to rasterize the interior of the implicit ADF glyph in a
//              separate step.
//
//          (c) After the distances in the distance buffer have been mapped to
//              density values as described above in step (7), the adjusted CSM
```

```
//              inside and outside filter cutoff values are restored to their
//              original values.
//
//      Implicit ADF rendering of a stylized stroke-based glyph (i.e, an SSF glyph)
//      uses a hybrid of the approaches used for rendering outline-based glyphs and
//      uniform-width stroke-based glyphs. Rendering an SSF glyph comprises the
//      following steps:
//
//          (1) Same as above
//          (2) Same as above
//          (3) Same as above
//
//          (4) Same as above with the following additions: a) corners are detected
//              in the SSF glyph and annotated in the SSF internal path; b) unit
//              normal vectors are computed from the original stroke path at each
//              pen position in the SSF internal path (these unit normal vectors are
//              used for generating implicit ADF boundary cells); c) the normalized
//              length along each stroke skeleton is computed for each pen position
//              in the SSF internal path (the normalized length is used for
//              evaluating profiles).
//
//          (5) First, stroke bodies are rendered as signed distance fields and
//              composited with distances of the previously rendered stroke bodies by
//              taking the maximum distance value (i.e., by performing a CSG union
//              operation). For each element of the SSF internal path, an implicit
//              ADF boundary cell (i.e., a line cell) is generated. The geometry of
//              the line cell is determined from its corresponding line segment and
//              unit normal vectors and is constructed to enclose all of the sample
//              points that may be closest to the line segment and are either inside
//              the stroke body or within the filter radius of the edges of the
//              stroke body. For each sample point in the line cell, the minimum of
//              the closest signed distances to the left and right edges of the
//              stroke body is determined from the line segment and the left and
//              right stroke profiles of the SSF glyph; this minimum distance is then
//              composited with the corresponding distance in the distance buffer by
//              taking the maximum distance value. Second, each endcap and corner
//              (i.e., each stroke serif) of the SSF glyph is rendered into a small
//              temporary distance buffer which is then composited with the main
//              distance buffer using a CSG union operation (i.e., by taking the
//              maximum distance value for each sample point). For each serif, the
//              serif's open path is transformed (i.e., rotated, translated, and
//              scaled) to fit the end or corner of the appropriate stroke skeleton,
//              and used to generated an implicit ADF of type ADF_OUTLINE_PATH whose
//              ADF coordinates are identical to image coordinates (i.e., the
//              rendering transform from ADF coordinates to image coordinates is the
//              identity matrix). An internal path of type ADF_OUTLINE_PATH, in which
//              curve segments of the implicit ADF are approximated by a set of line
//              segments, is then created from the implicit ADF. The internal path is
//              rendered by first generating an implicit ADF boundary cell for each
//              element of the internal path, determining an unsigned distance from
//              each sample point inside the ADF boundary cell to the corresponding
//              element, and compositing the negative of the unsigned distance with
//              the corresponding distance in the temporary distance buffer by taking
//              the maximum negative value. The serif's open path is then closed
//              (e.g., by connecting each end of an open corner path to the corner
//              point for a stroke corner) and the sign of the distance value of each
//              sample point inside the closed path is changed from negative to
//              positive.
//
//          (6) Similar to uniform-width stroke-based glyphs, the interior is
//              rendered during step (5), thereby eliminating the need to rasterize
//              the interior of the SSF glyph in a separate step.
//
//          (7) Same as above
//
//      Support for implicit ADF SAZ alignment zone detection consists of a function
//      for sampling the distance field of a specified implicit ADF to determine a
```

```
//      distance map of size w x h, i.e., a 2D image of floating point distance
//      values. During sampling, the [0.0,1.0] x [0.0,1.0] bounding box of the
//      implicit ADF is scaled to the [0,w-1] x [0,h-1] distance map.
//
//      Implicit ADF validation provides a function for drawing a specified implicit
//      ADF and its SAZ alignment zones. This function permits validation of the
//      distance field of the implicit ADF glyph and for the placement of its SAZ
//      alignment zones. Note that MAZ alignment zones are not supported by this
//      function.
//-----------------------------------------------------------------------------------
```