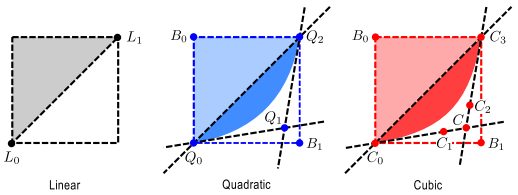


**Figure 3:** *Implicit representation for abstract segments. The figure shows the top-right configuration of figure 2. Other configurations are analogous.*

**Linear segments** Let vector $\boldsymbol{s} = \begin{bmatrix} s_x & s_y & 1 \end{bmatrix}^T$ hold the homogeneous coordinates of the sample. The equation for line $L_0\,L_1$ can be written in the form $\boldsymbol{k}\,\boldsymbol{s} = 0$ for an affine function (i.e., a row vector) $\boldsymbol{k} = \begin{bmatrix} a & b & c \end{bmatrix}$. It is easy to pick $\boldsymbol{k}$ such that $\boldsymbol{k}\,\boldsymbol{s} > 0$ for samples to the left of $L_0L_1$. We use this implicit line test to decide on which side of a linear segment each sample lies.

Loop and Blinn [2005] gave a neat procedure for generalizing this implicit test for quadratic and cubic Béziers. They use a result by Salmon [1852] that ensures it is always possible to find affine functions $\boldsymbol{k}$, $\boldsymbol{\ell}$, $\boldsymbol{m}$ such that the implicit tests become

$$\text{integral quadratic:} \quad (\boldsymbol{k}\,\boldsymbol{s})^2 > \boldsymbol{l}\,\boldsymbol{s} \tag{9}$$

$$\text{rational quadratic:} \quad (\boldsymbol{k}\,\boldsymbol{s})^2 > (\boldsymbol{l}\,\boldsymbol{s})(\boldsymbol{m}\,\boldsymbol{s}) \tag{10}$$

$$\text{integral cubic:} \quad (\boldsymbol{k}\,\boldsymbol{s})^3 > (\boldsymbol{l}\,\boldsymbol{s})(\boldsymbol{m}\,\boldsymbol{s}). \tag{11}$$

Each abstract segment stores the row vectors corresponding to the required affine functions, and use them to quickly perform implicit tests on the required samples. Unfortunately, monotonization is not enough to guarantee the correctness of the implicit test inside the entire bounding box.

**Quadratic segments** Consider the quadratic in figure 3. The quadratic could reenter triangle $Q_2B_0Q_0$ (shaded in light blue) after leaving the bounding box. This would lead to the wrong classification of sample points. Loop and Blinn [2005] use the GPU rasterizer to generate fragments only inside triangle $Q_0Q_1Q_2$. This would require 3 additional implicit line tests. Instead, it is sufficient to restrict the test to triangle $Q_0B_1Q_2$, using the bounding box and a single implicit test against segment $Q_0Q_2$.

*Proof:* The quadratic cannot cross segment $Q_0Q_2$ outside of points $Q_0$ and $Q_1$ since it can intersect a straight line at most twice. Similarly, it cannot cross segments $Q_0Q_1$ and $Q_1Q_2$. Indeed, since the curve is tangent at both $Q_0$ and $Q_2$, these points already count as two intersections. Finally, note the quadratic cannot intersect segments $Q_0B_1$ and $B_1Q_2$ without first incurring forbidden additional intersections with segment $Q_0Q_1$ or $Q_1Q_2$.

**Cubic segments** Cubics are more complicated. To prevent the curve from looping back and intersecting segment $C_0C_3$ (and the curve itself), Loop and Blinn [2005] split cubics at a double-point whenever one is found for $t_d$ with $0 < t_d < 1$. This requires solving a simple quadratic equation, and we do the same. The test is now valid inside the convex-hull of the control polygon (Loop and Blinn [2005] rasterize two triangles to generate the appropriate pixels). In our case, this would require at least four implicit line tests and some book keeping. Instead, we split the cubics at an inflection point whenever one is found for $t_i$ with $0 < t_i < 1$. This requires solving a simple quadratic but guarantees the control polygon is convex, and therefore the intersection of lines $C_0C_1$ and $C_2C_3$ happens at a point $C$ inside the bounding box. It is sufficient to restrict the implicit test to the triangle $C_0CC_3$.

*Proof:* We again use root-counting arguments. First, we show the curve cannot intersect segment $C_0C_3$. If it did, it would either have to exit triangle $C_0CC_3$ again through segment $C_0C_3$ (but it cannot have 4 intersections with line $C_0C_3$), or would have to self-intersect (but by assumption it has no double-point for $t \in (0, 1)$). The arguments for why the curve cannot intersect segment $C_0C$ and $CC_3$ are analogous, so consider segment $C_0C$. We start from the part of the curve that exits triangle $C_0CC_3$ at $C_0$. If it is below $C_0C$, then $C_0$ is an inflection and exhausts all 3 possible intersections with line $C_0C$. If it is above $C_0C$, then $C_0$ is only a tangent. Now recall the curve cannot intersect $C_0C_3$. Therefore, in order to intersect $C_0C$ a third time, it would have to either go up around triangle $C_0CC_3$, thereby intersecting $CC_3$ four times (twice at the tangent $C_3$ and twice before it can reach $C_0C$), or go down back into $C_0C$ (wasting the third and last intersection with line $C_0C$ at a point outside of segment $C_0C$). Now consider the part of the curve exiting at $C_3$. If it exits to the right of $CC_3$, then $C_3$ is an inflection and precludes the fourth intersection with line $CC_3$, needed to reach segment $C_0C$. If it exits to the left, it would intersect line $C_0C$ the third time outside of segment $C_0C$, since it cannot intersect segment $C_0C_3$.

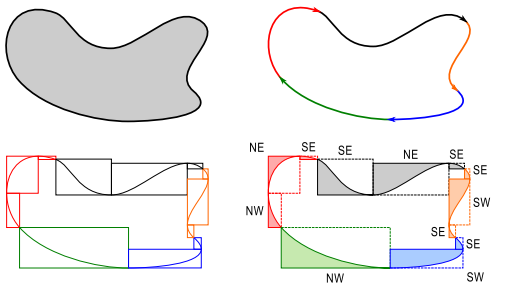


**Figure 4:** *The segments in each path are classified and decomposed into monotonic* abstract segments. *Abstract segments can be queried for their bounding box, their orientation* (NE,NW,SE,SW)*, and for the side on which a sample lies.*

In the remainder of the paper, we assume that all segments have been previously monotonized and we manipulate them as abstract entities. These abstract segments can be queried for a bounding box, for an orientation (NE,NW,SE,SW), and for the side on which a given sample lies. Figure 4 illustrates the decomposition of a filled primitive into its abstract segments.

## 4 The shortcut tree

Assume we have partitioned the illustration area into a union of small regions. The key strategy for speeding up the inside-outside test within each region is to cull segments that can never be intersected by rays originating from the region in the $+x$ direction. It is immediately clear that we can eliminate an entire segment whenever its bounding box is completely above, below, or to the left of the window. The additional insight from the work of Nehab and Hoppe [2008] is that we need only include the segments that actually intersect the window. Their *lattice-clipping* algorithm generates a regular grid of cells, each containing only the parts of segments that overlap with them. In addition to the clipped segments, Nehab and Hoppe include winding increments and auxiliary segments that ensure the correct number of intersections is computed for any sample inside each cell. These are the *shortcuts* in our shortcut tree. We improve on that idea by considering all segments in parallel and by creating an adaptive tree structure rather than a regular grid. Furthermore, we include complete segments into the cells, rather than cutting segments into pieces. This allows us to avoid computing expensive intersections between segments and cell boundaries, and reduces the total number of primitives in the tree.