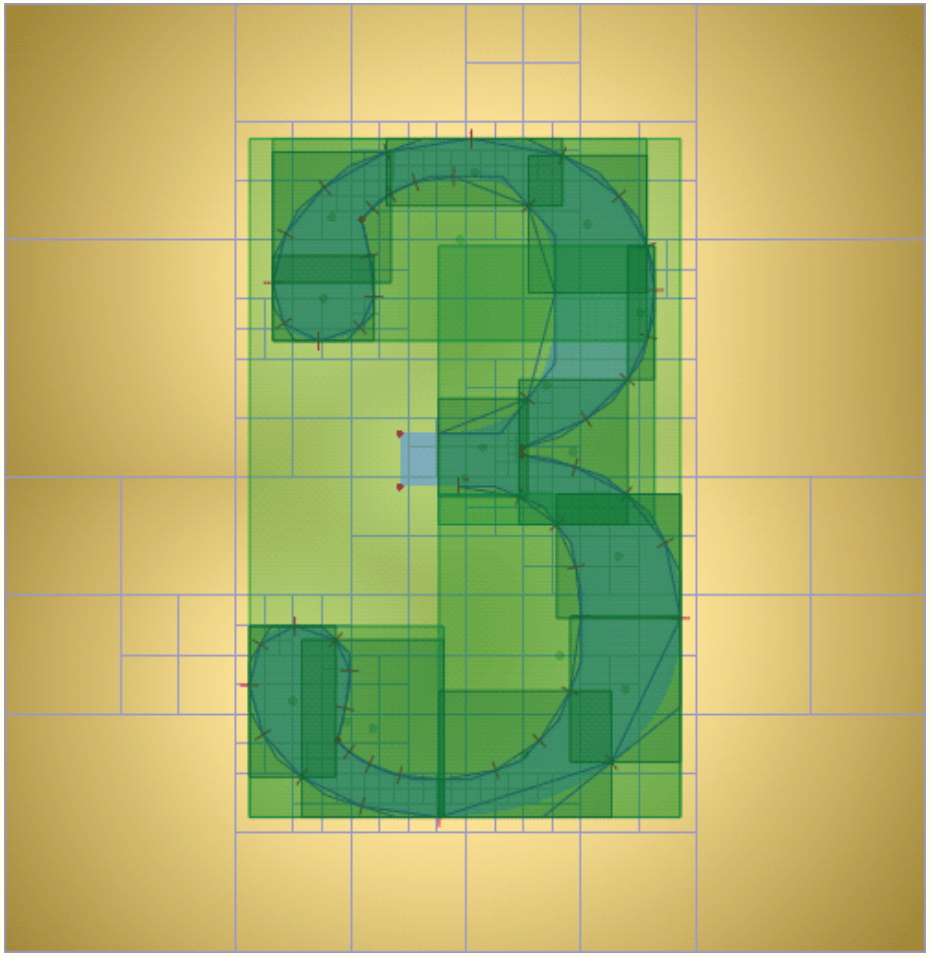
Efficient Distance Field Computation Using Cluster Trees by Elena Jakubiak

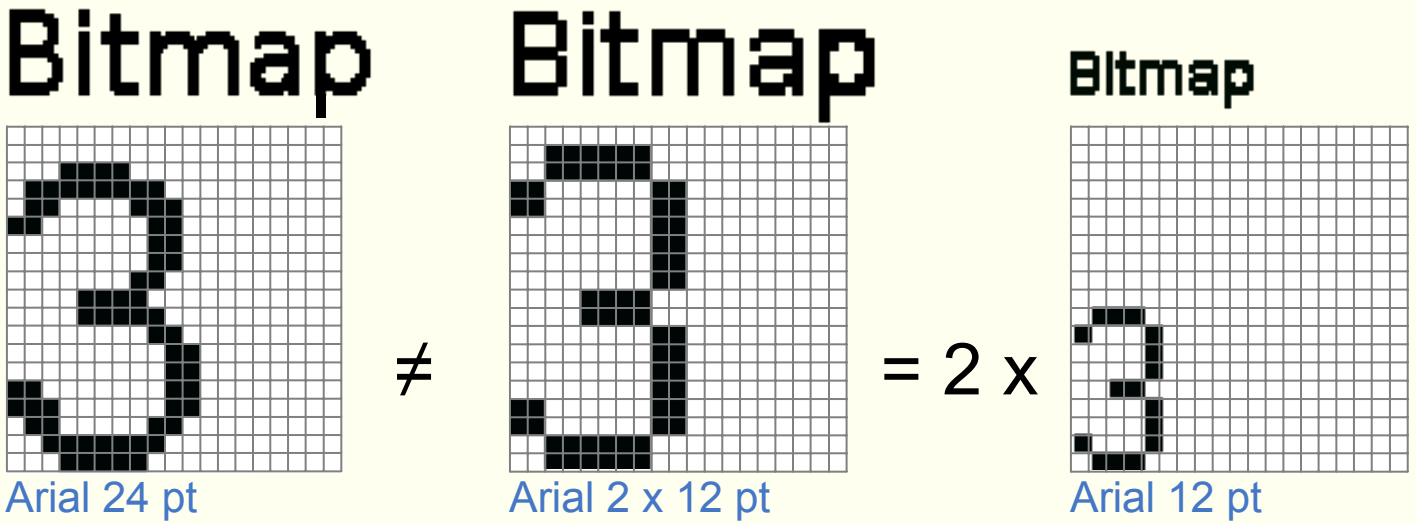
- Background
  - Fonts
  - Distance Fields
- ADF Font Rendering
  - Overview
  - Timing Problem
  - Possible Solutions
  - Cluster Trees
- Conclusions & Future Work



Fonts

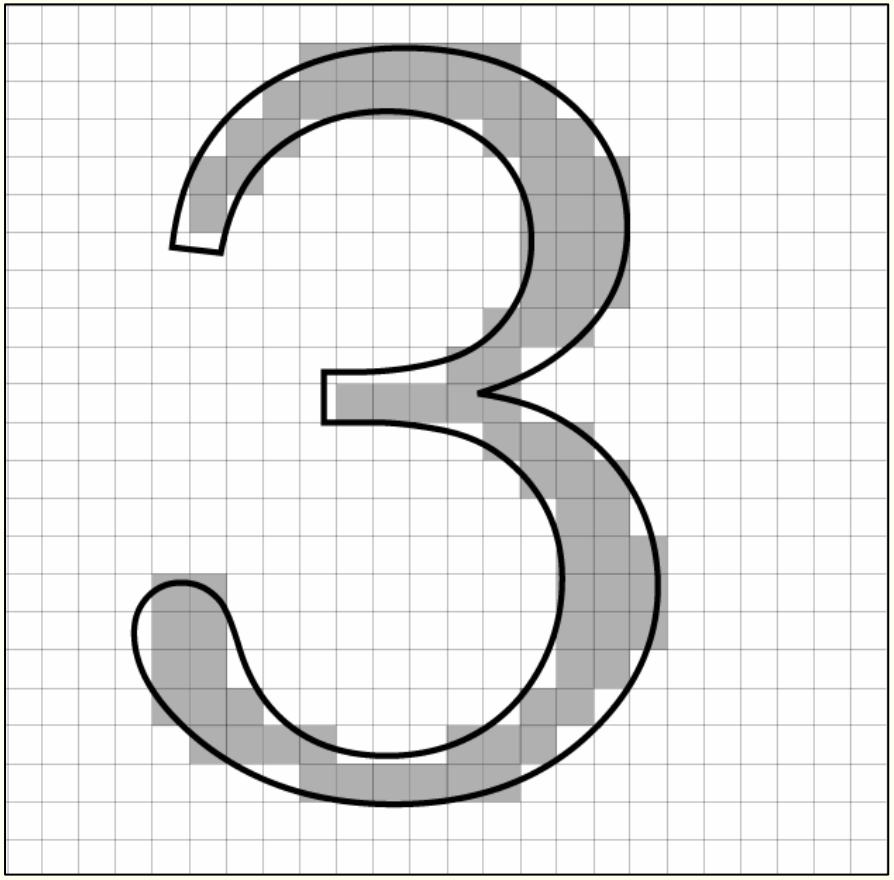
Bitmaps

- Original digital font
- Not scalable → Stores one bitmap for each size-type combination
- Sufficient when only a few fonts are needed
- Digital publishing requires a variety of fonts



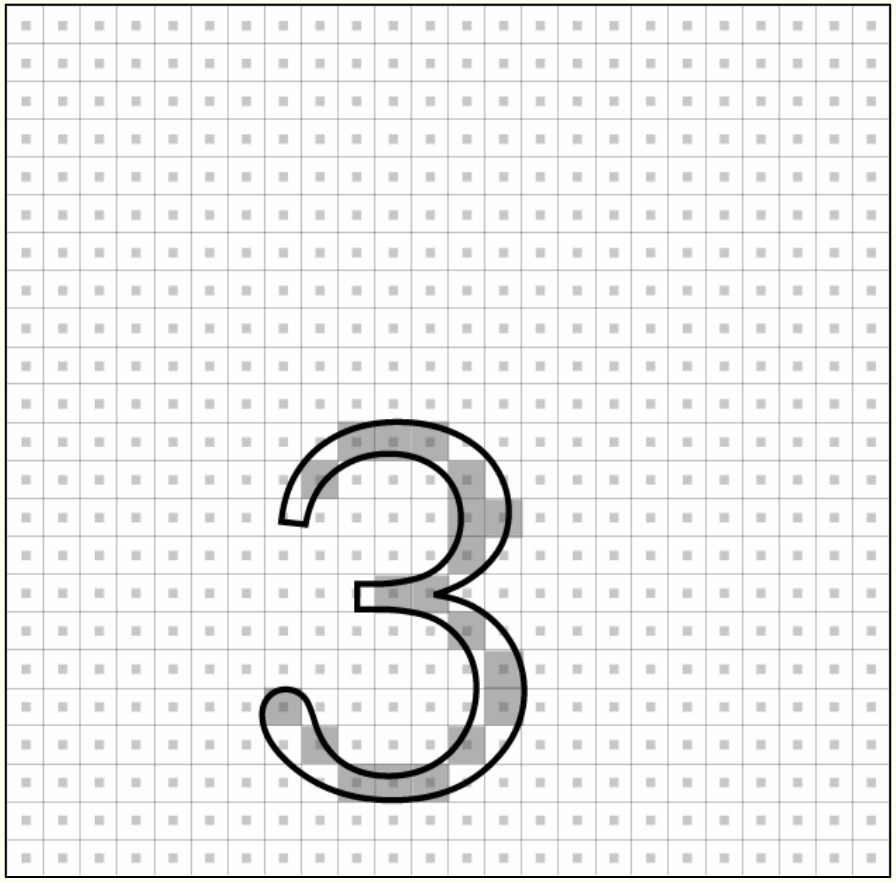
Outline Fonts

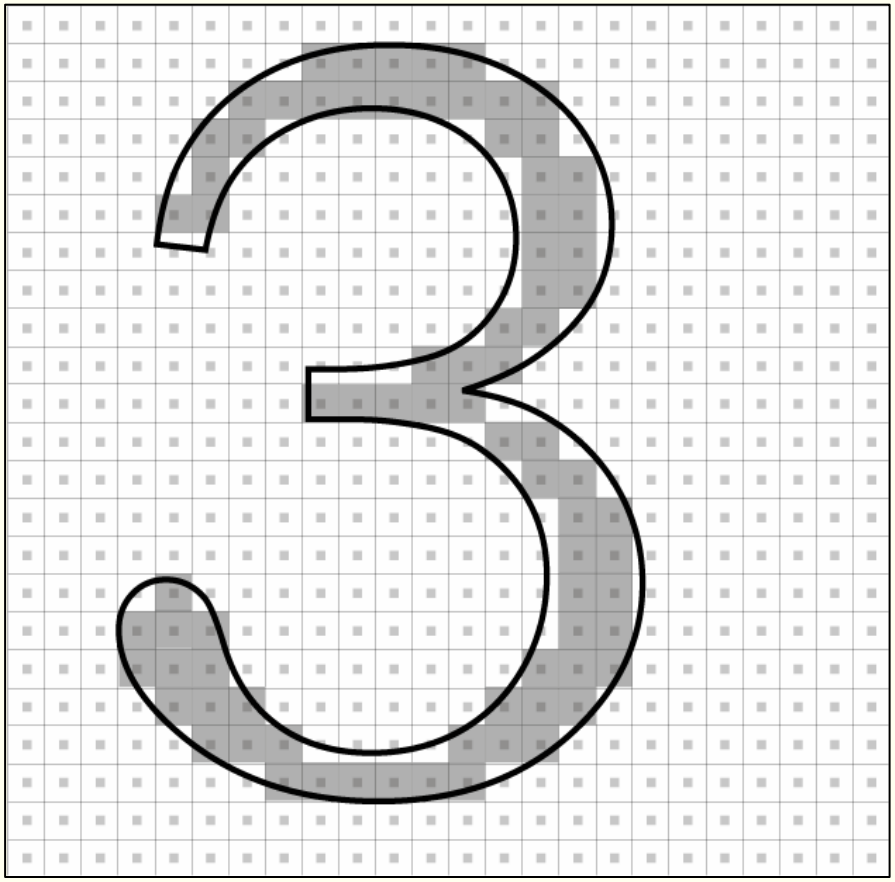
- Outline Fonts defined by a series of straight line segments and curves
- Use the outline to determine which pixels to color



Outline Fonts

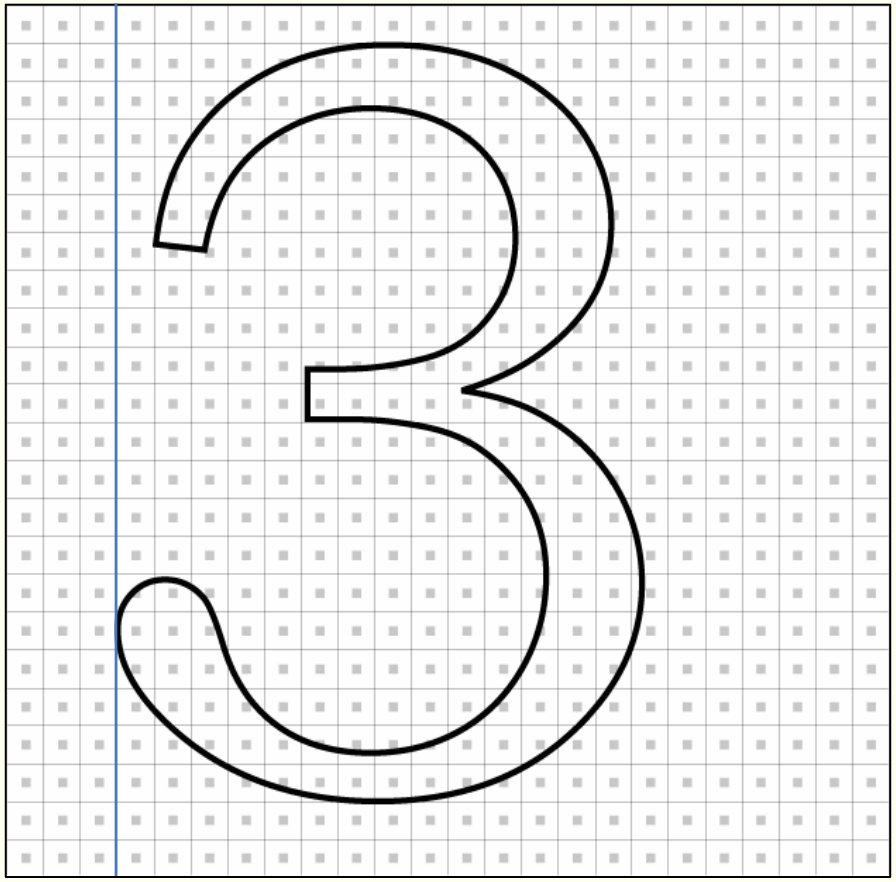
- Pros: scalable so one outline used for all size-type combinations
- Cons: complicated hinting & poor antialiasing

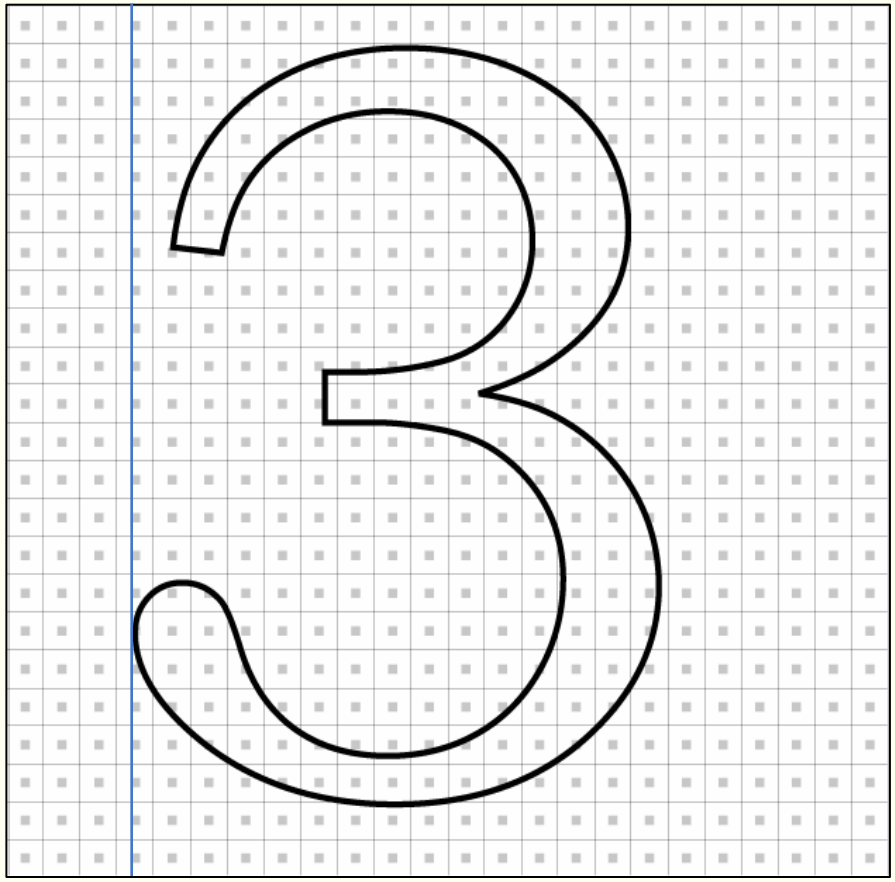




Hinting

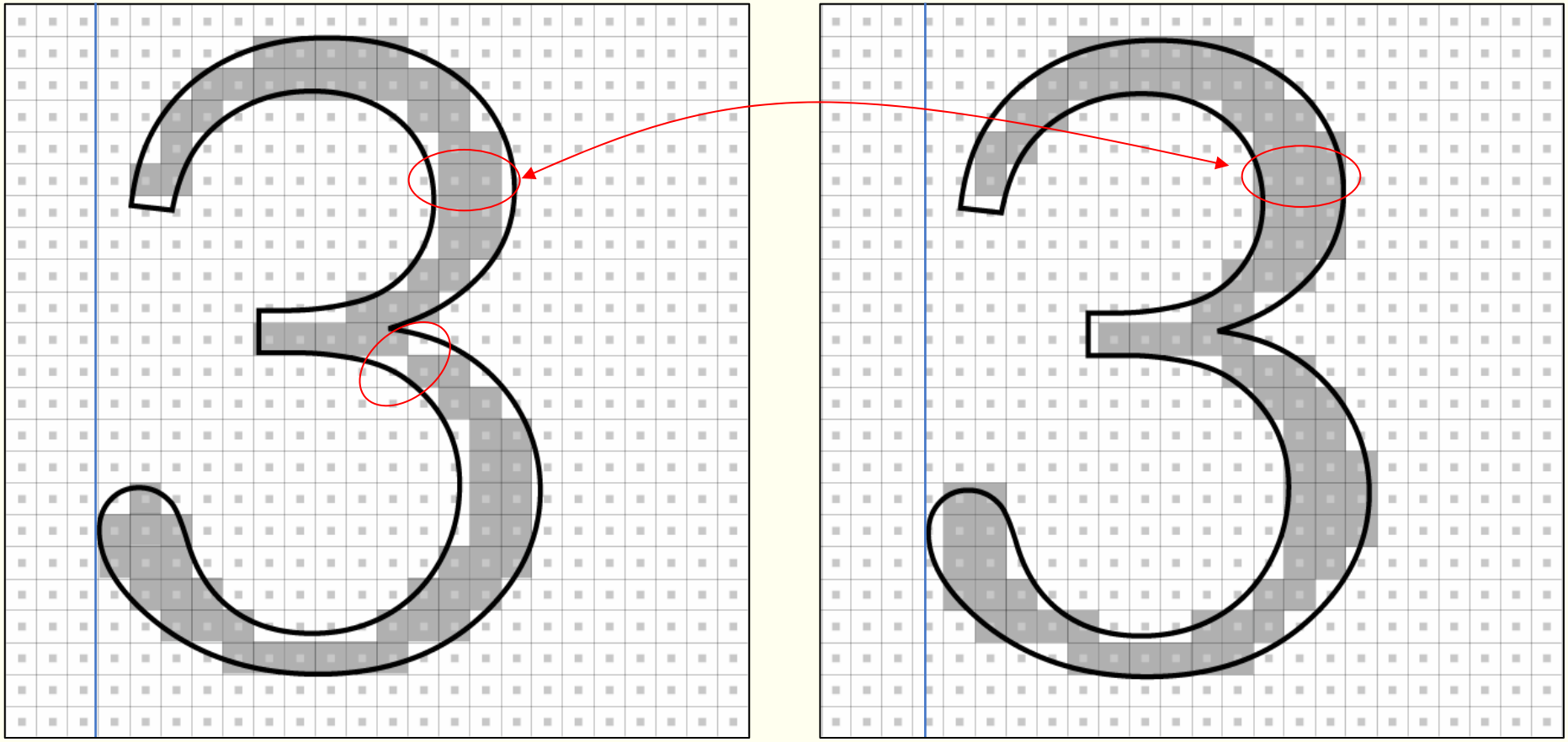
- Outlines may align differently relative to pixel boundaries based upon placement on the page





Hinting

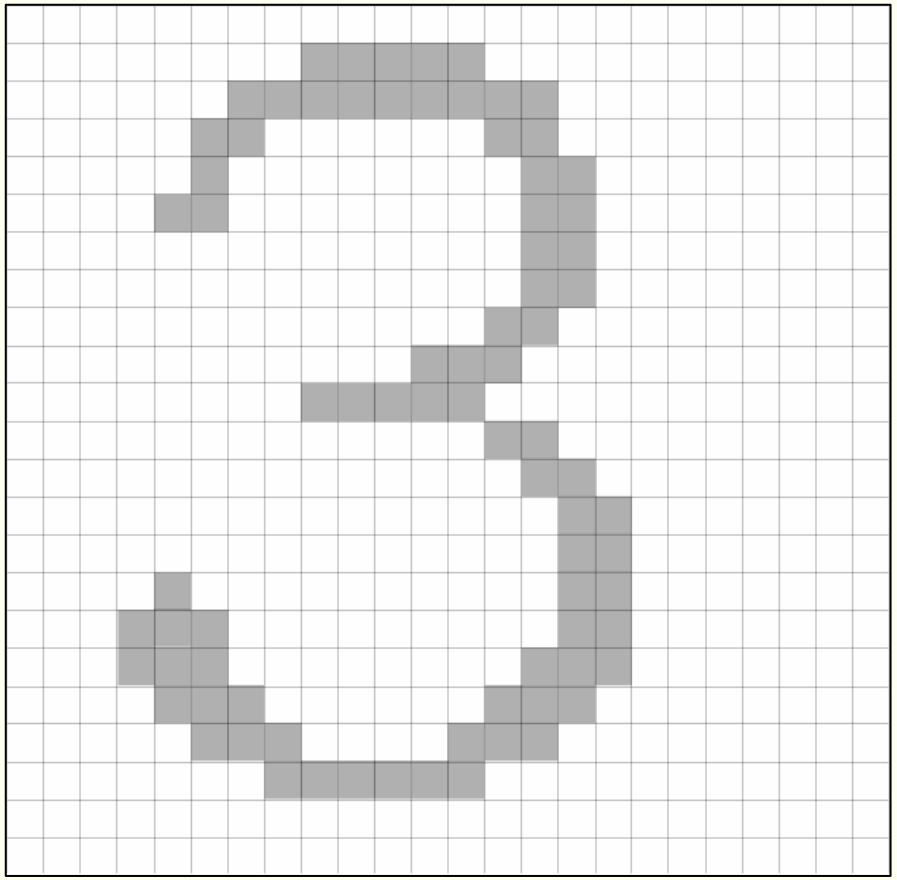
- Outlines may align differently relative to pixel boundaries based upon placement on the page
- The same glyphs render differently

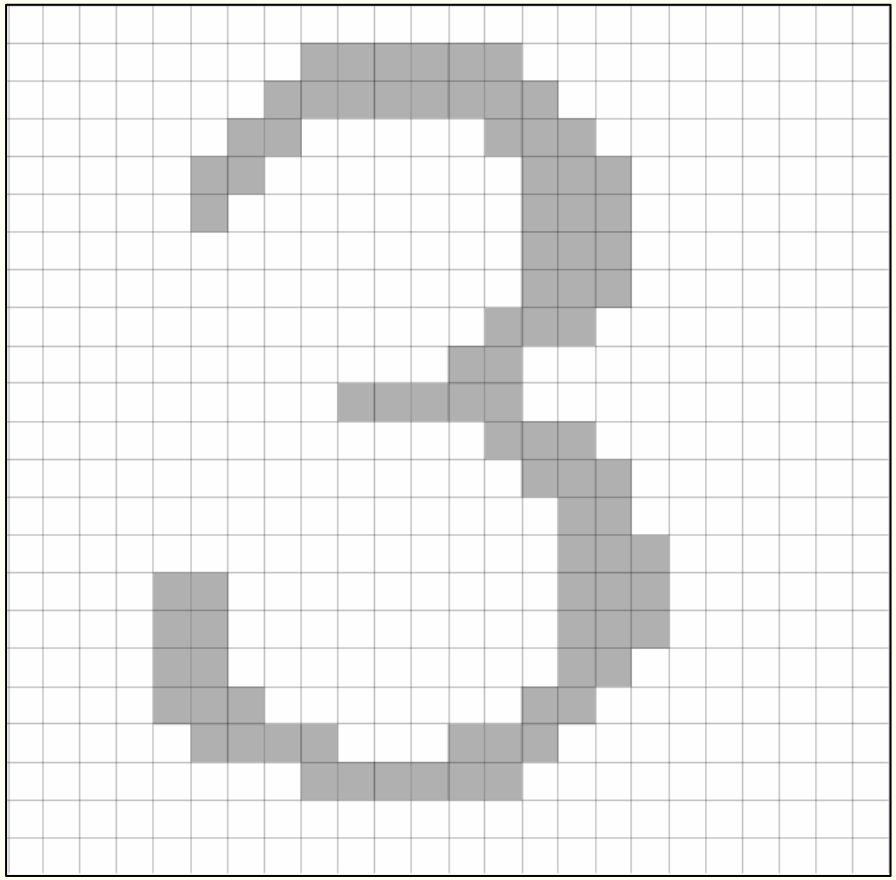


Hinting

- "Hints" are included to help fit the glyph to the grid.
- Hinting is very complicated and few people are fluent

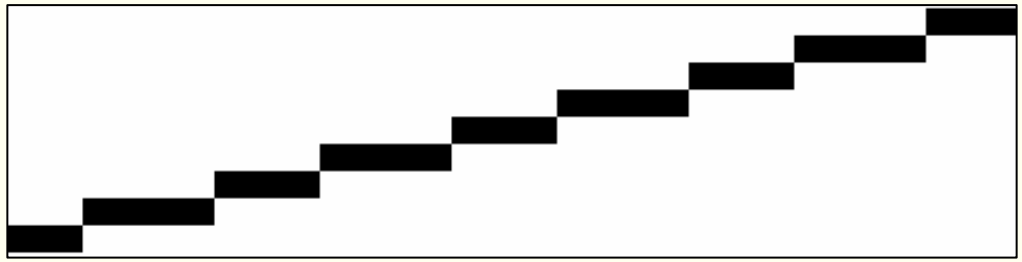
"This is Verdana, a font designed for maximum on-screen legibility and readability. Verdana was designed by world renowned type designer **Matthew Carter**, and hand-hinted by leading hinting expert, Agfa Monotype's **Tom Rickner**…." *http://www.microsoft.com/typography/cleartype/tuner/4.htm*





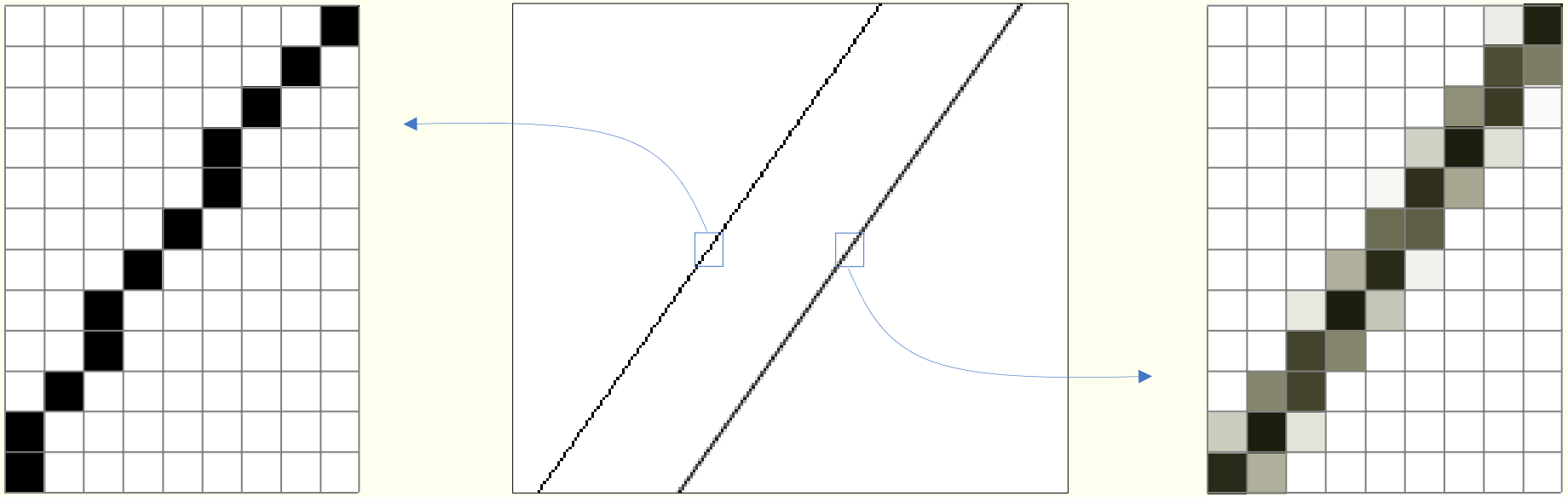
Aliasing

- Effect of approximating a continuous object with discrete samples
- In graphics, the appearance of jagged edges



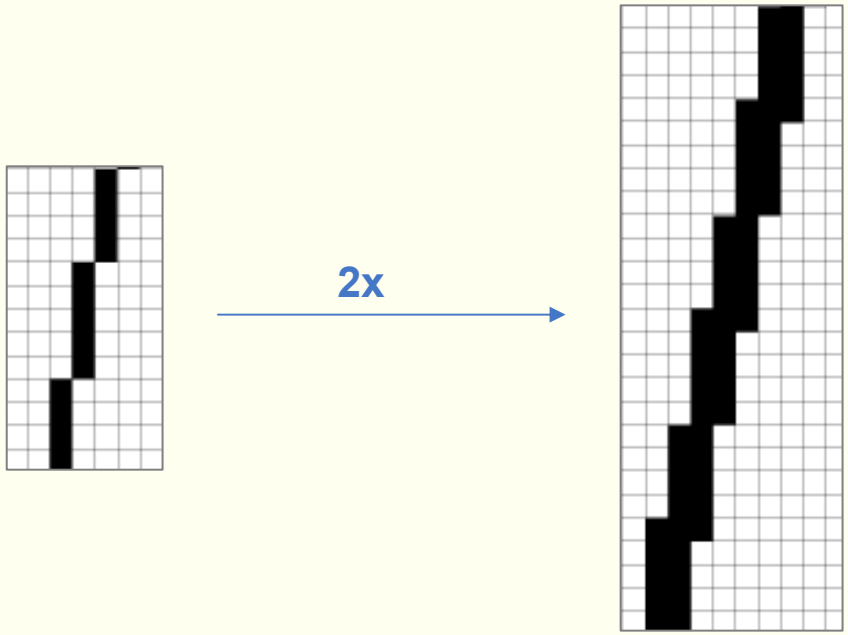
Antialiasing

- Remove jaggies by blurring edges



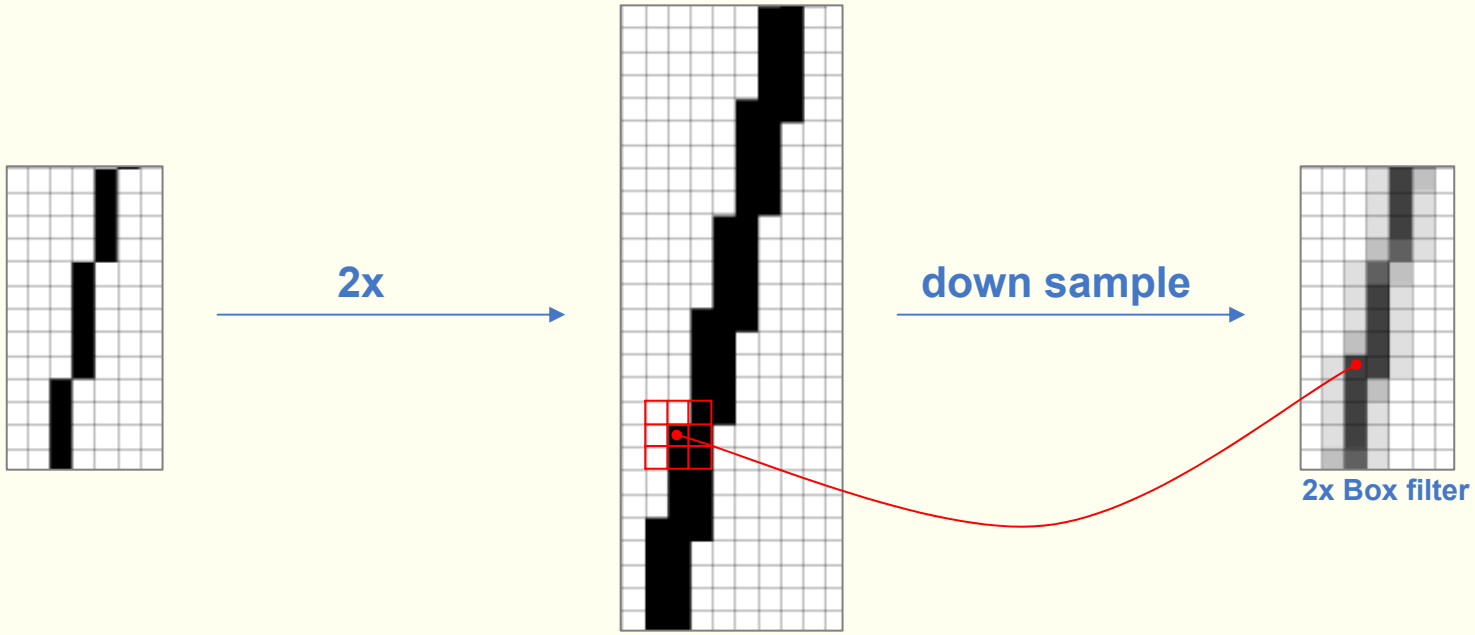
Super-sampling

- Glyphs currently antialiased using “Super-sampling”
- Super-sampled Lines:
  - Render line at a larger size (2x)



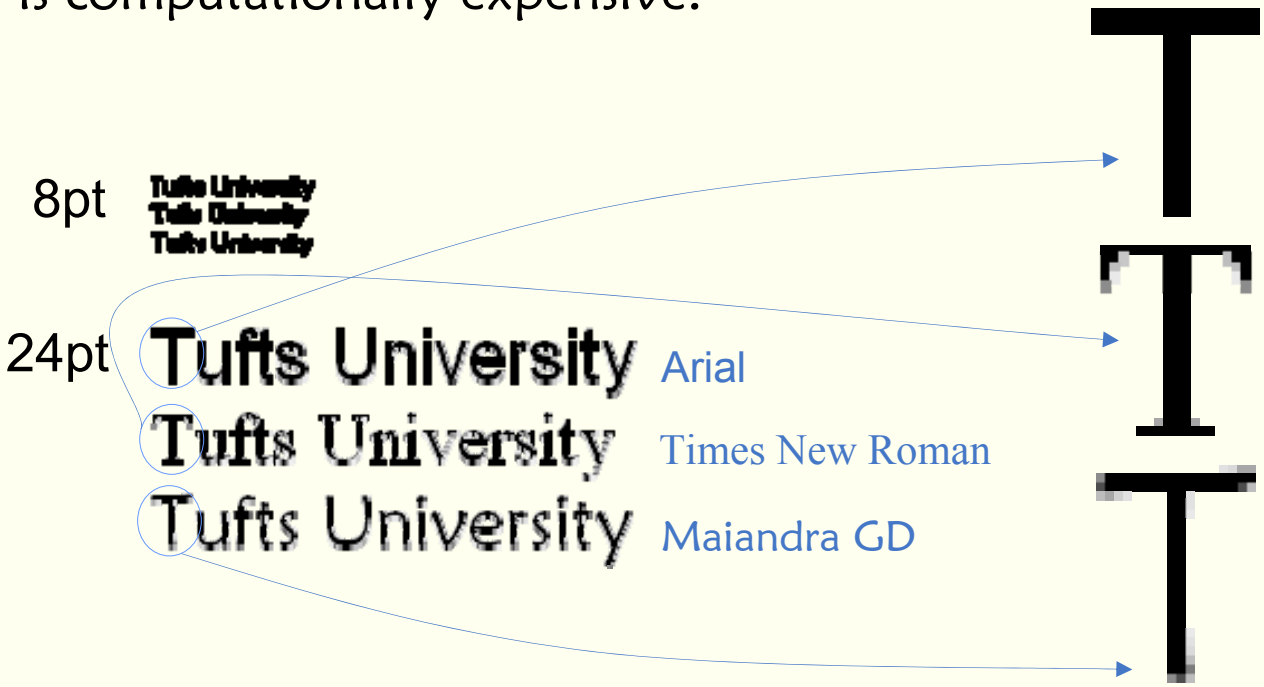
Super-sampling

- Glyphs currently antialiased using “Super-sampling”
- Super-sampled Lines:
  - Render line at a larger size (2x)
  - To determine the color for a pixel in the smaller line, compute a weighted average of its neighbors in the larger image.



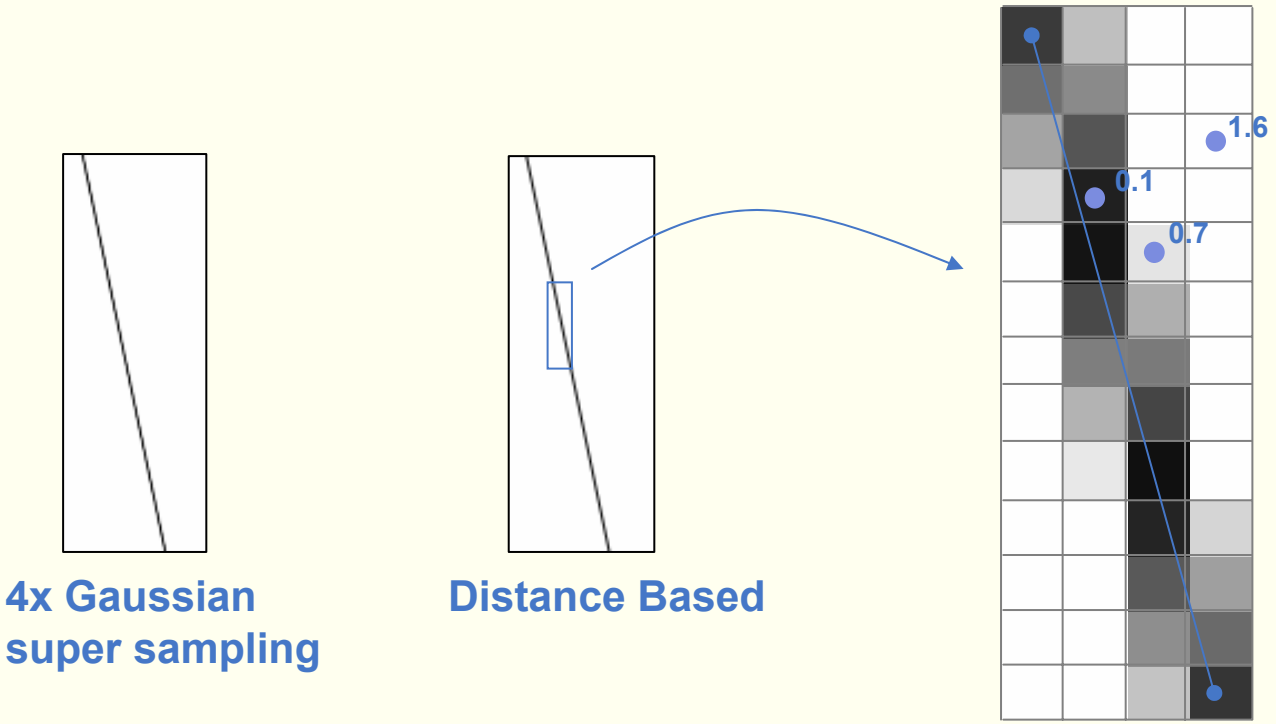
Poor Antialiasing

- Poor antialiasing
  - Excessively blurry
  - Inconsistent rendering
  - Difficult to read, especially at smaller sizes
- To get better results with super-sampling, need to take more samples but this is computationally expensive.



Distance-Based Antialiasing

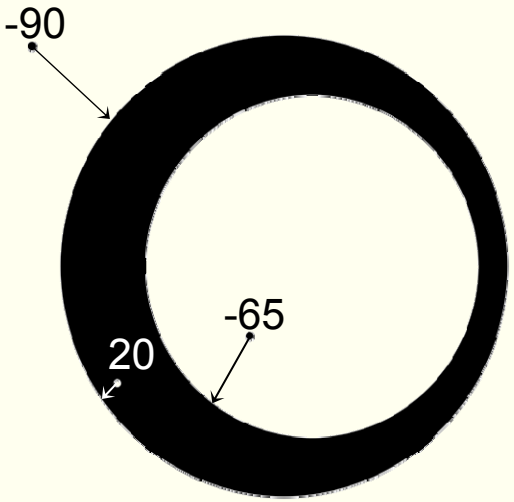
- Compute grayscale values using distance from the pixel's center to the line
- Can extend this technique to rendering fonts



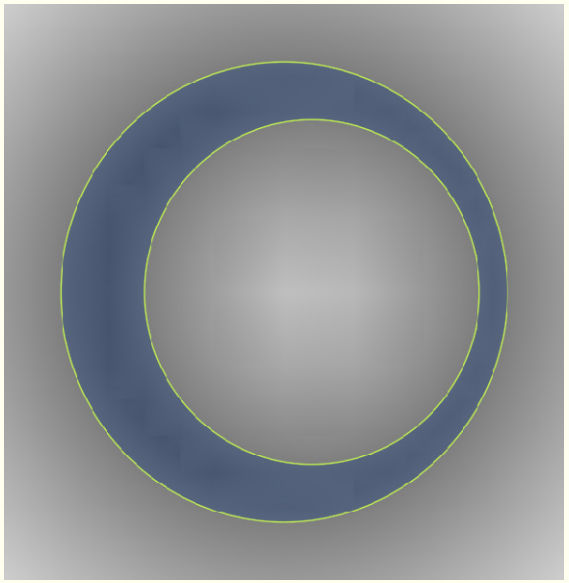
Distance Fields

A Better Approach: Distance Fields

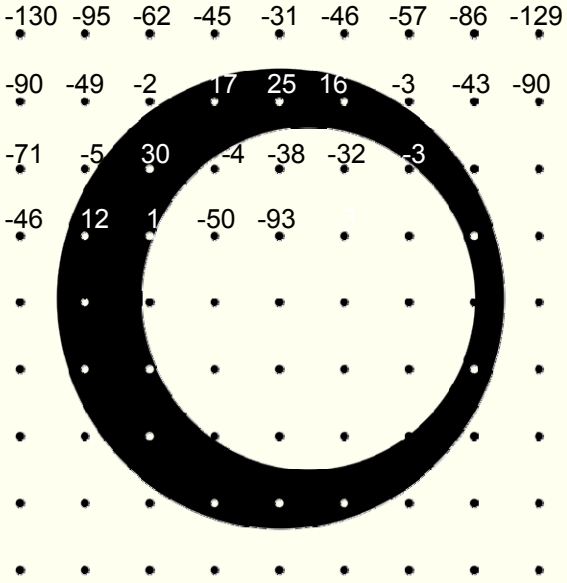
- **Distance Field:** a scalar field representing a shape **S** with boundary **dS** that specifies the distance to **dS** from any point in the field.



2D shape with sampled distances to the surface



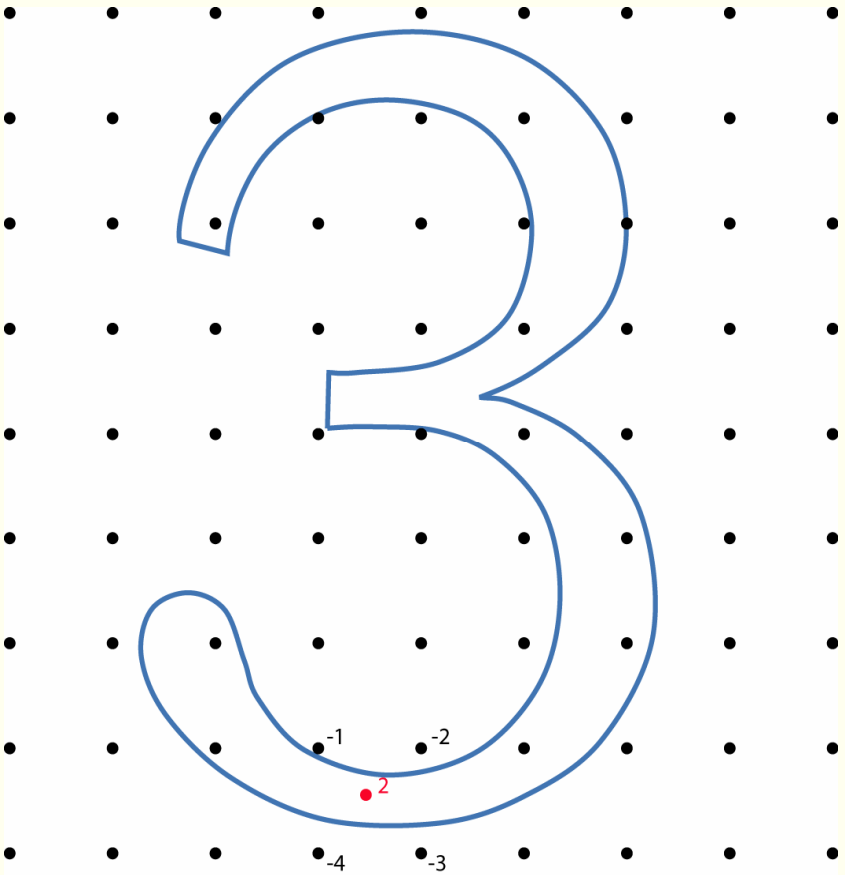
2D distance field: Color based on distance from boundary

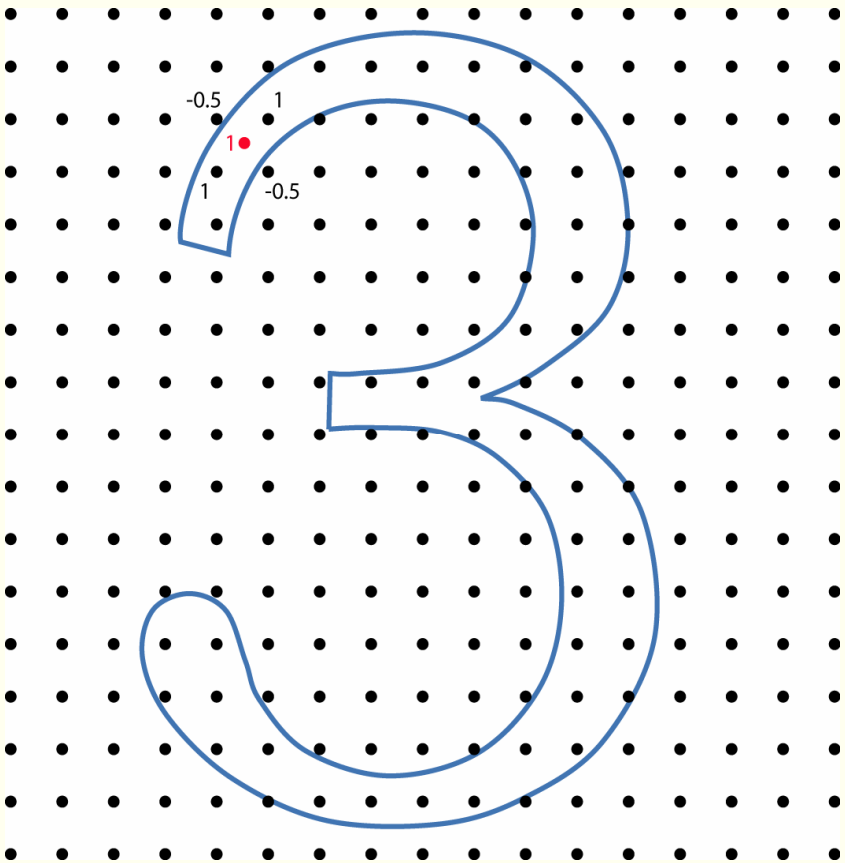


Regularly sampled distance values: reconstruct distance field using interpolation

A Better Approach: Distance Fields

- Use sampled distances to represent a glyph
- How many samples are sufficient?



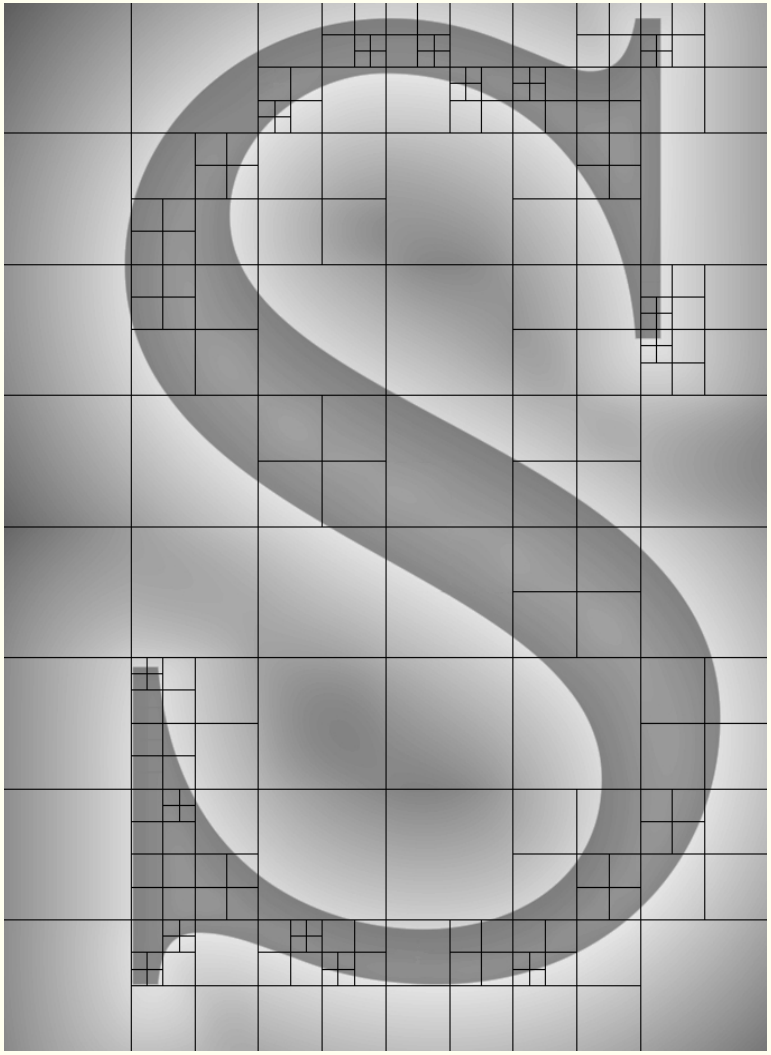


Adaptively Sampled Distance Fields

- **Adaptively Sampled Distance Fields** (ADFs) provide an efficient means for representing and processing distance fields.

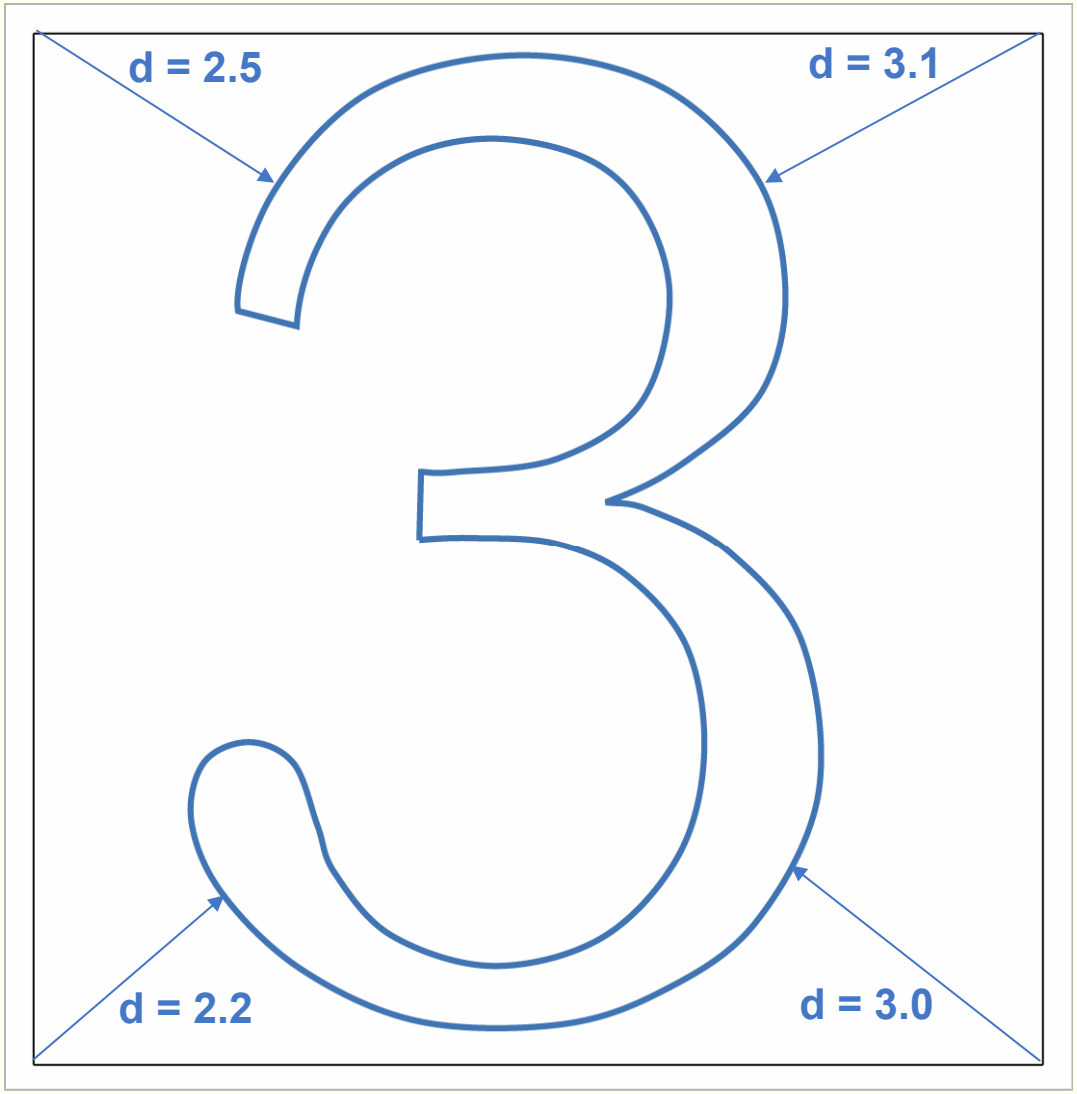
- Don't sample at regular intervals
- Start with a few samples
- If the distance between sample points changes quickly, take more samples as needed

- The color of a pixel is determine by its distance to the glyph



Constructing an ADF

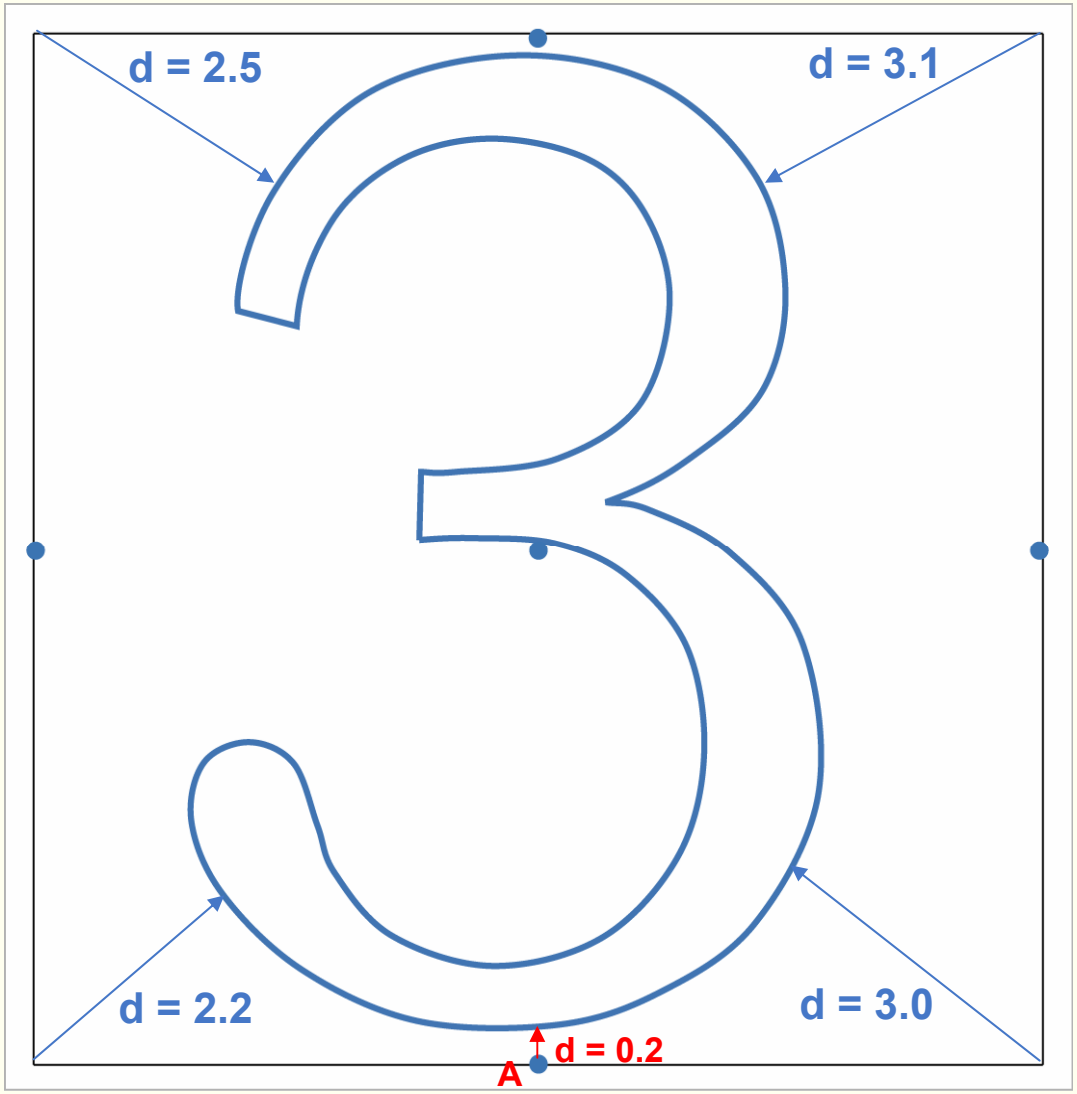
- Compute the distance to the glyph at the 4 cell boundaries



Constructing an ADF

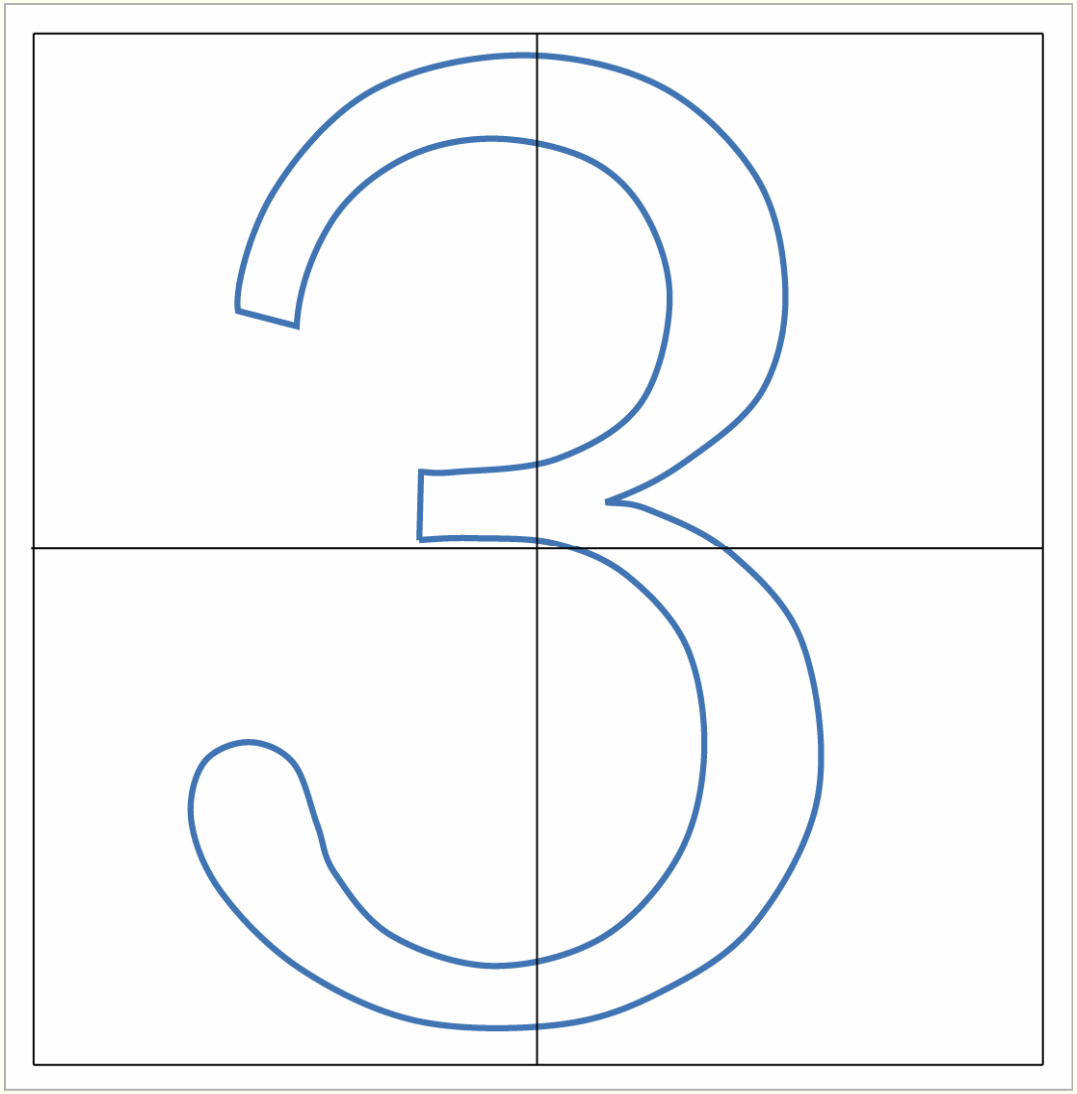
- At each test point (●)
  - Interpolate between cell corners to compute a reconstructed distance
  - Compare reconstructed distances to exact distances

- For point A, the distances are
  - Exact: d = 0.2
  - Reconstructed: d = 2.6



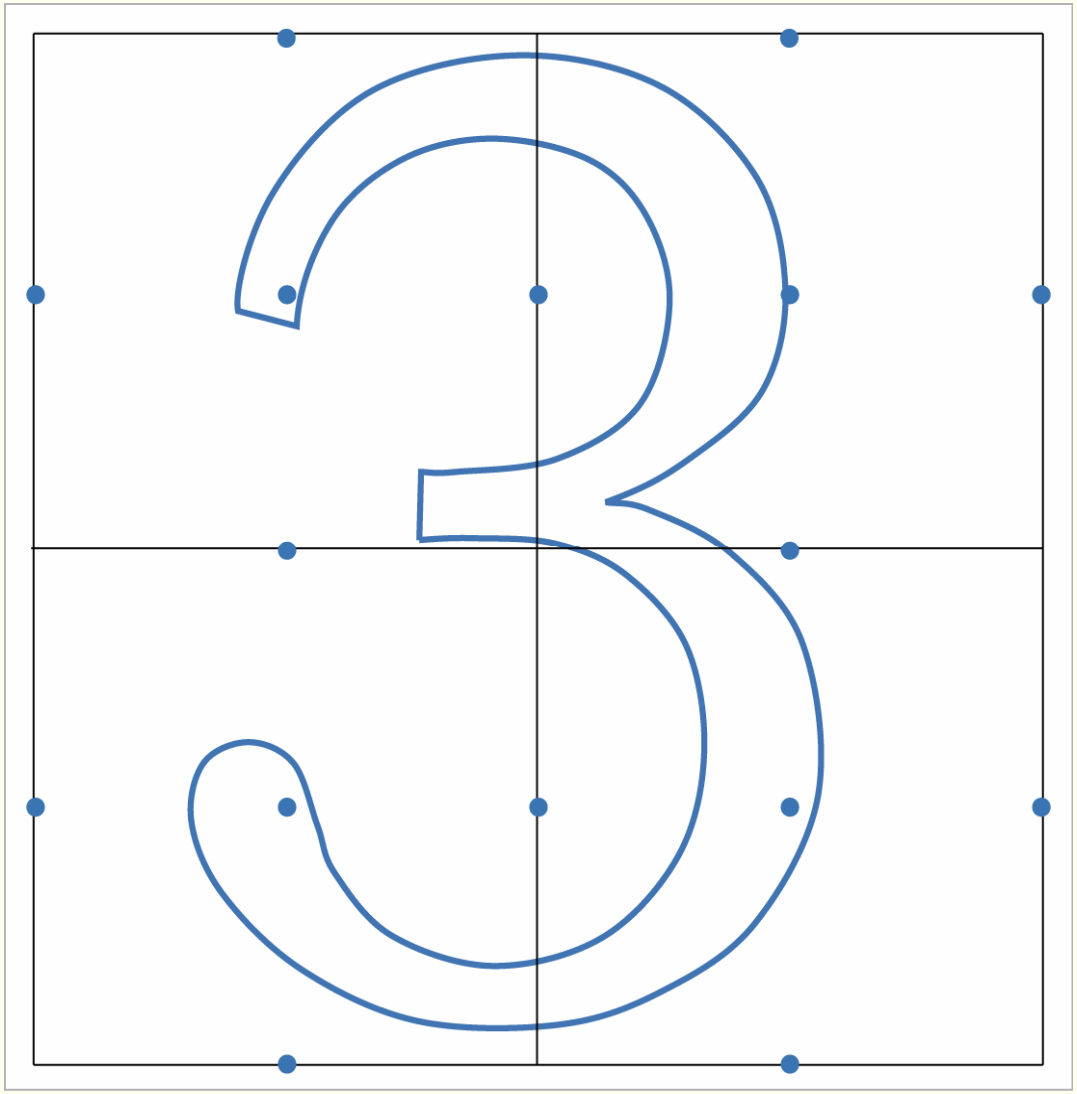
Constructing an ADF

- At each test point (●)
  - Interpolate between cell corners to compute a reconstructed distance
  - Compare reconstructed distances to exact distances
- For point A, the distances are
  - Exact: d = 0.2
  - Reconstructed: d = 2.6
- If reconstructed distances are not sufficiently accurate, subdivide the cell and repeat



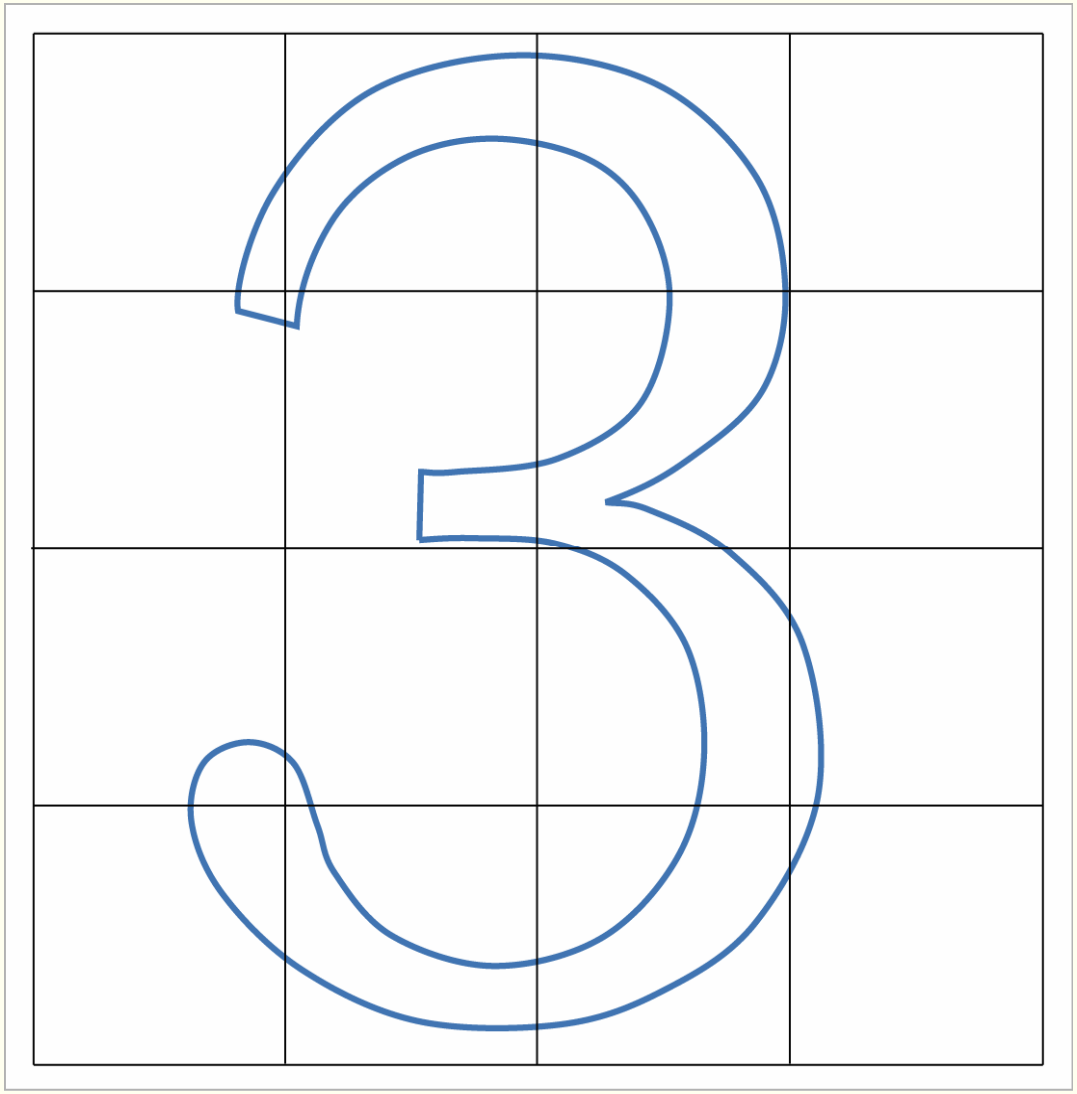
Constructing an ADF

- Compute and test reconstructed distances
- Subdivide as needed



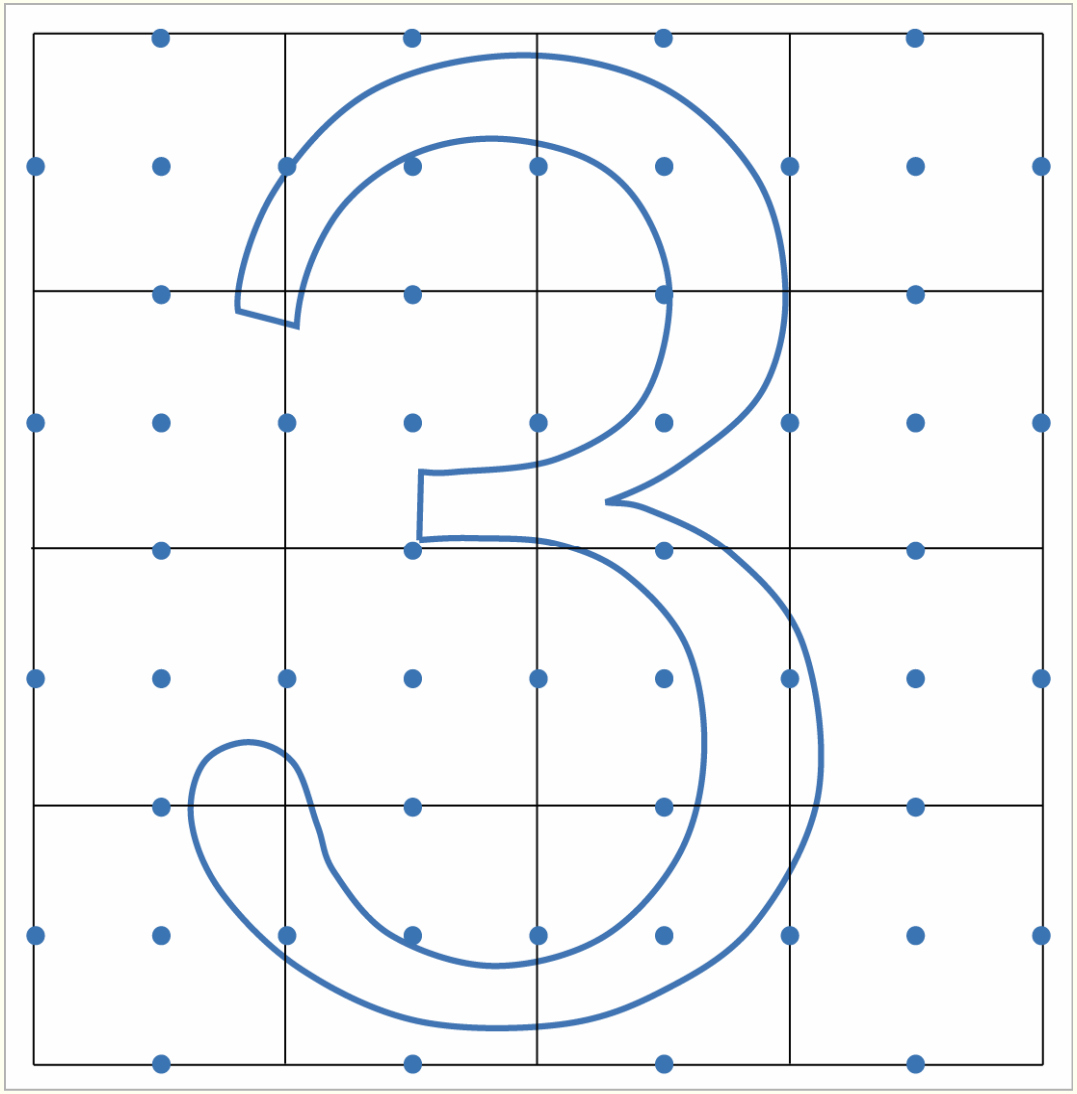
Constructing an ADF

- Compute and test reconstructed distances



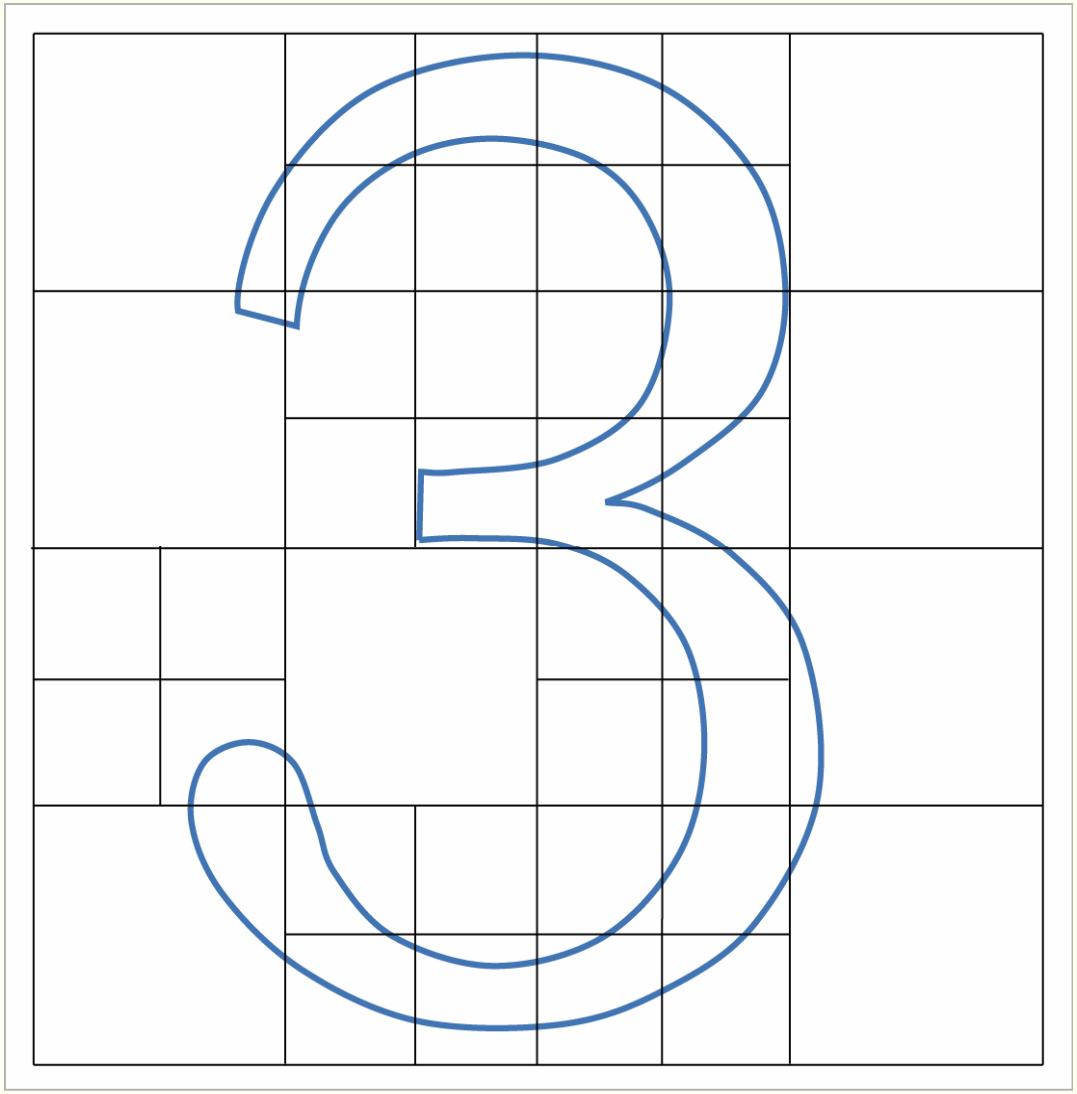
Constructing an ADF

- Compute and test reconstructed distances
- Subdivide as needed



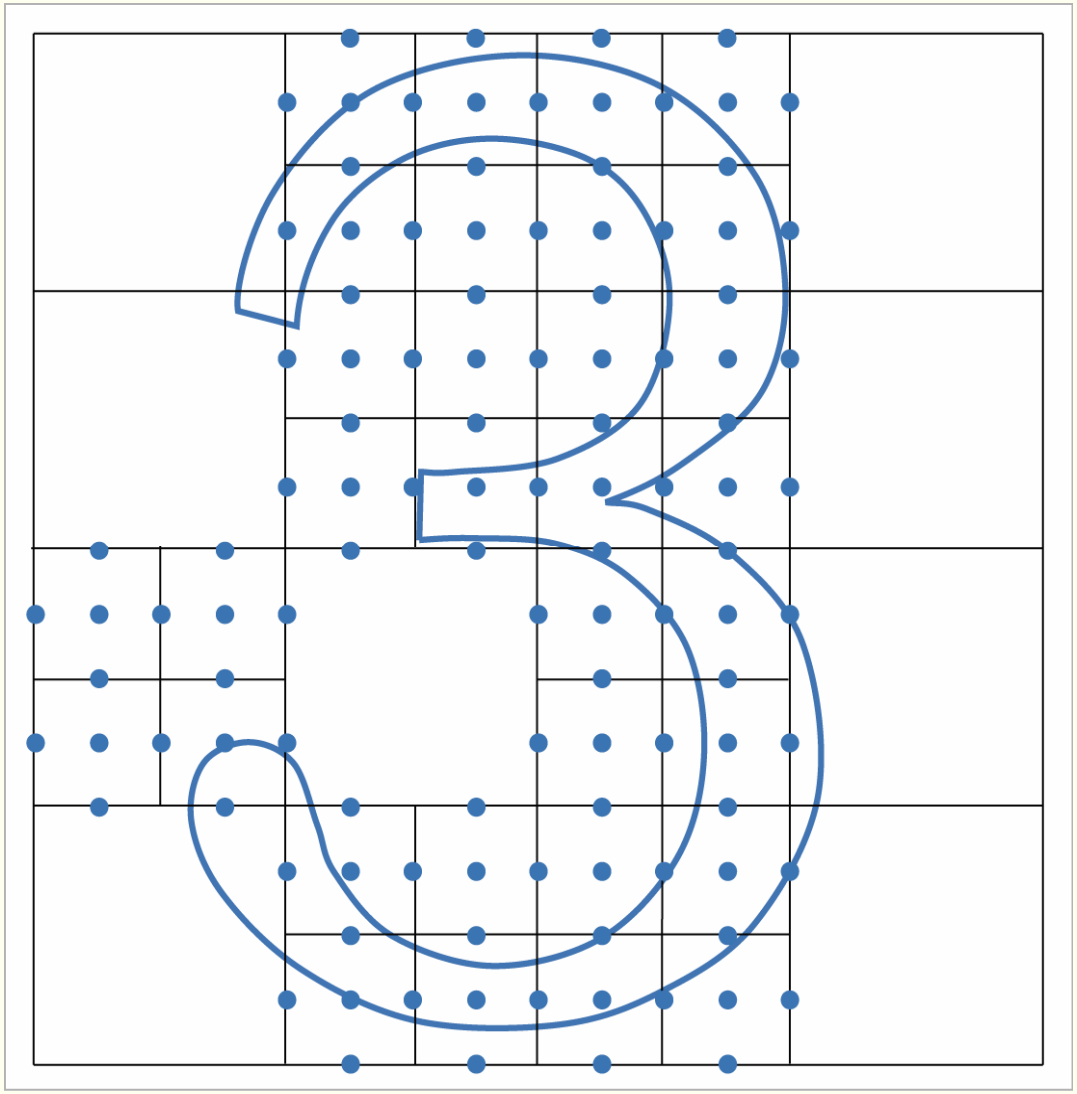
Constructing an ADF

- Compute and test reconstructed distances



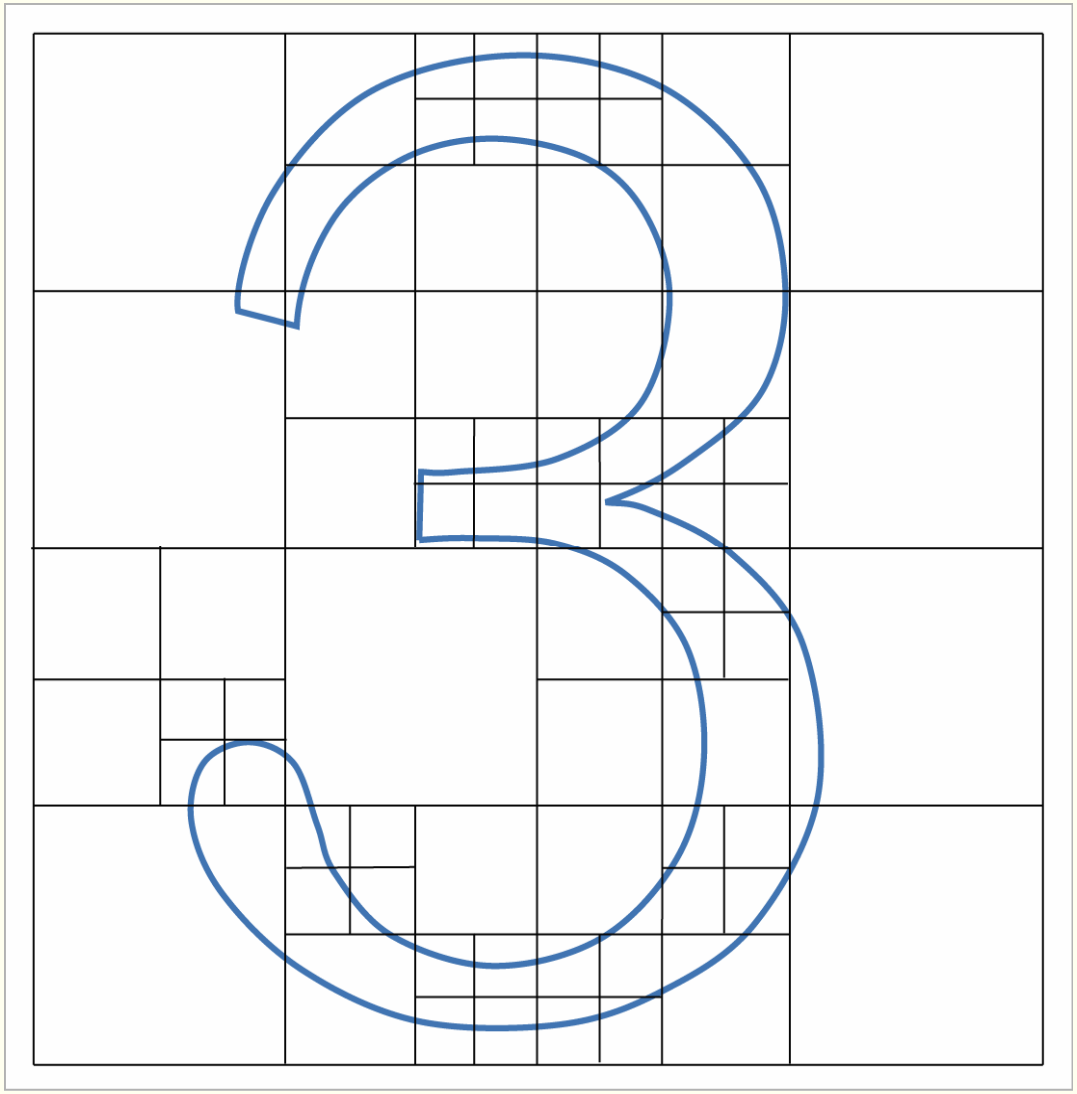
Constructing an ADF

- Compute and test reconstructed distances
- Subdivide as needed



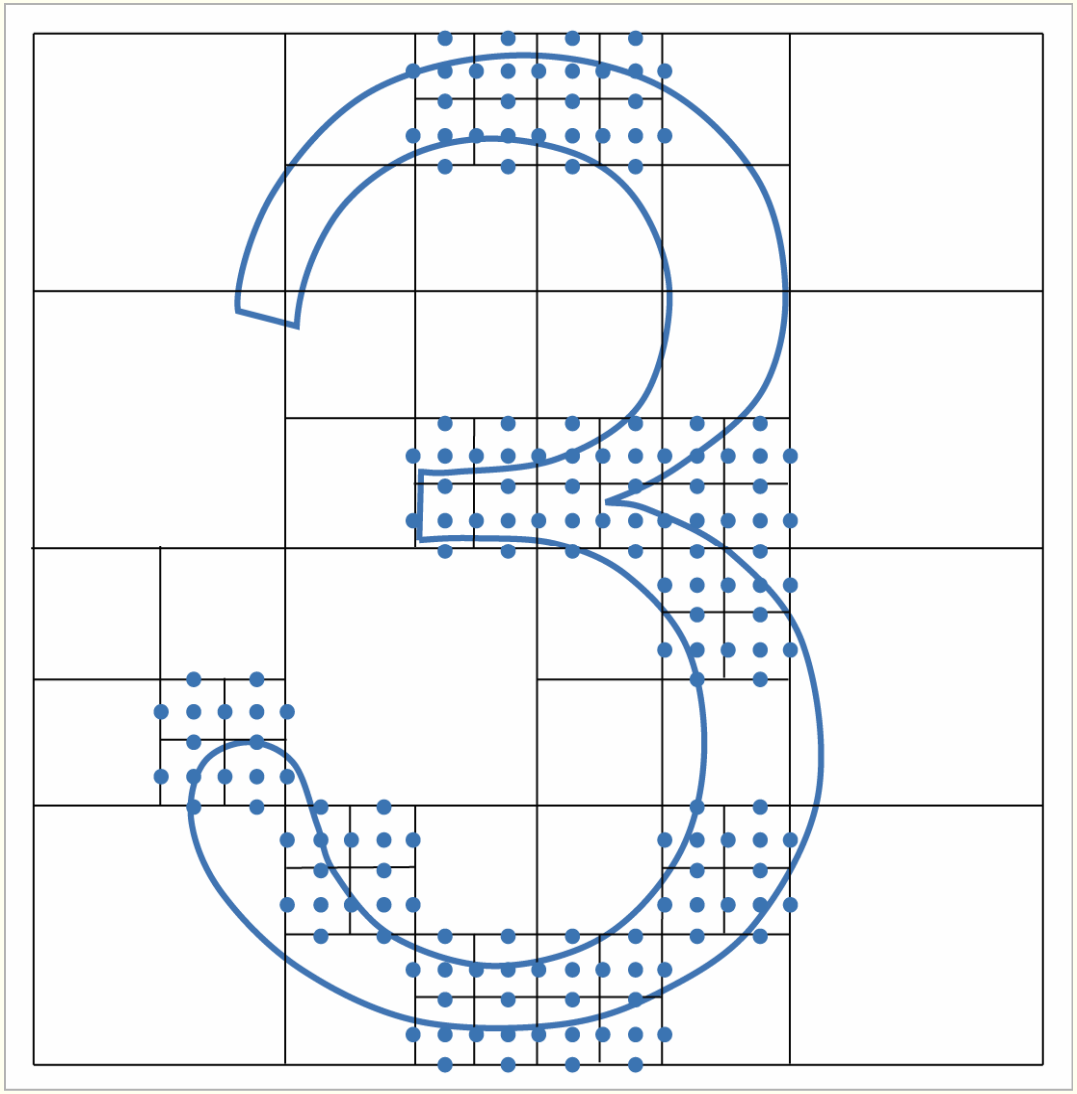
Constructing an ADF

- Compute and test reconstructed distances



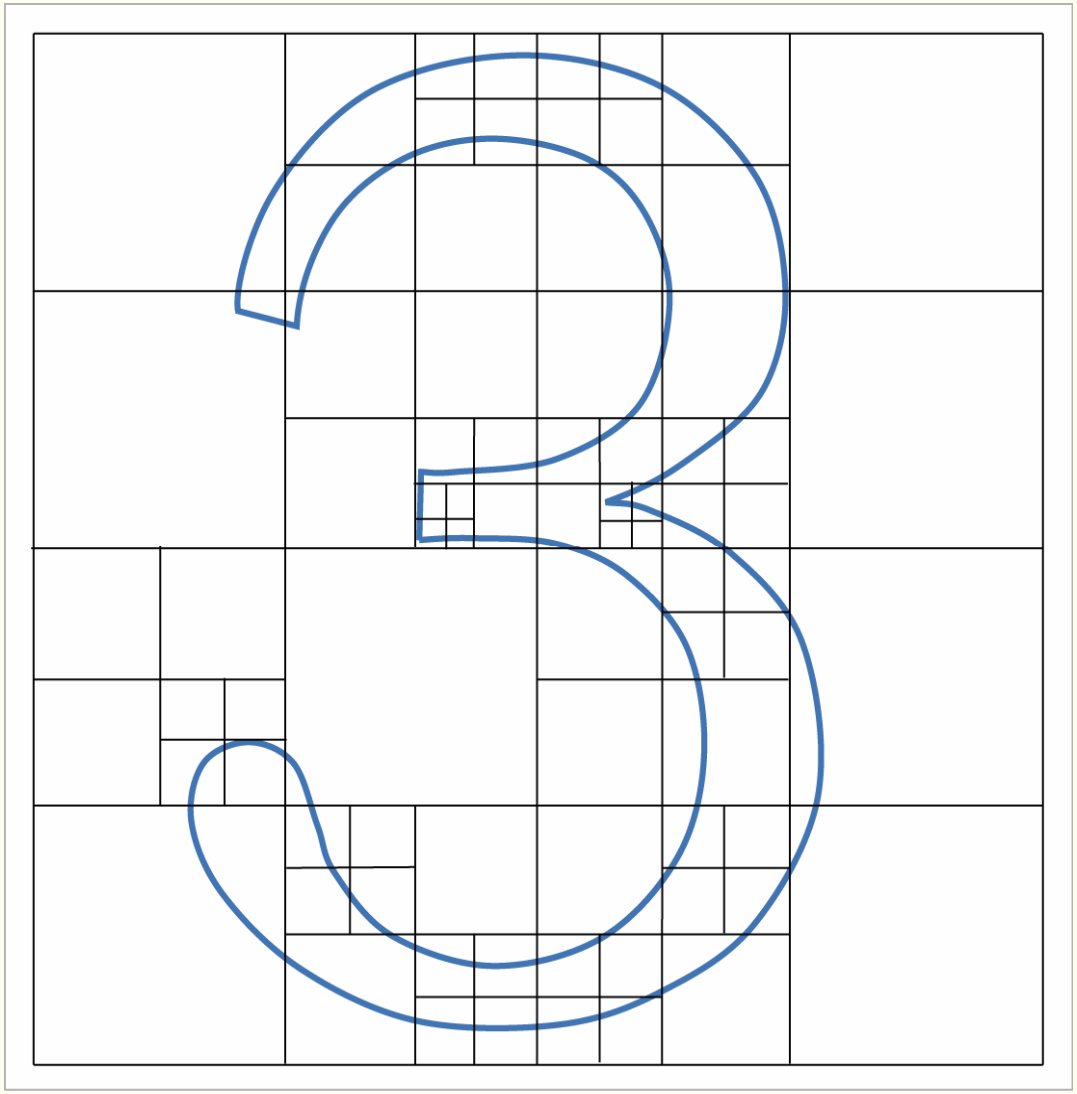
Constructing an ADF

- Compute and test reconstructed distances
- Subdivide as needed



Constructing an ADF

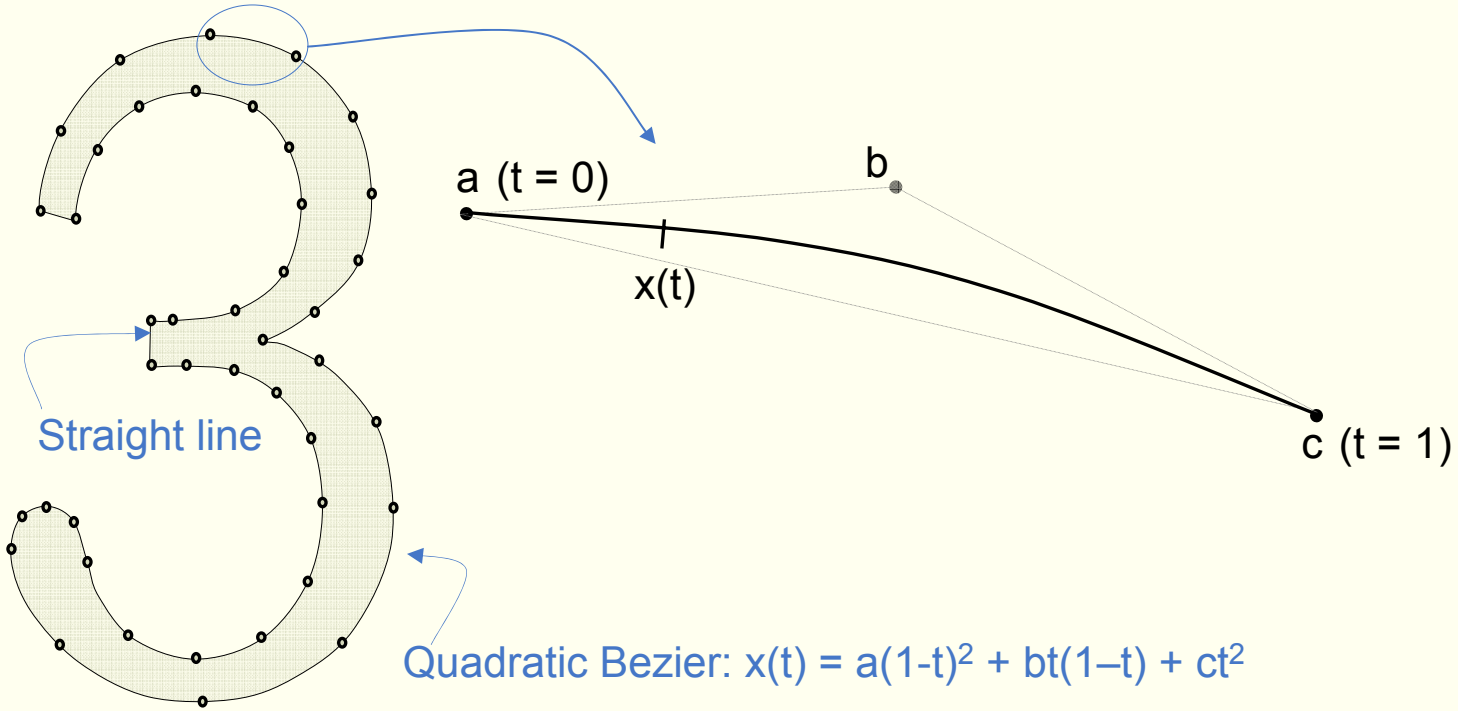
- Compute and test reconstructed distances



The Problem

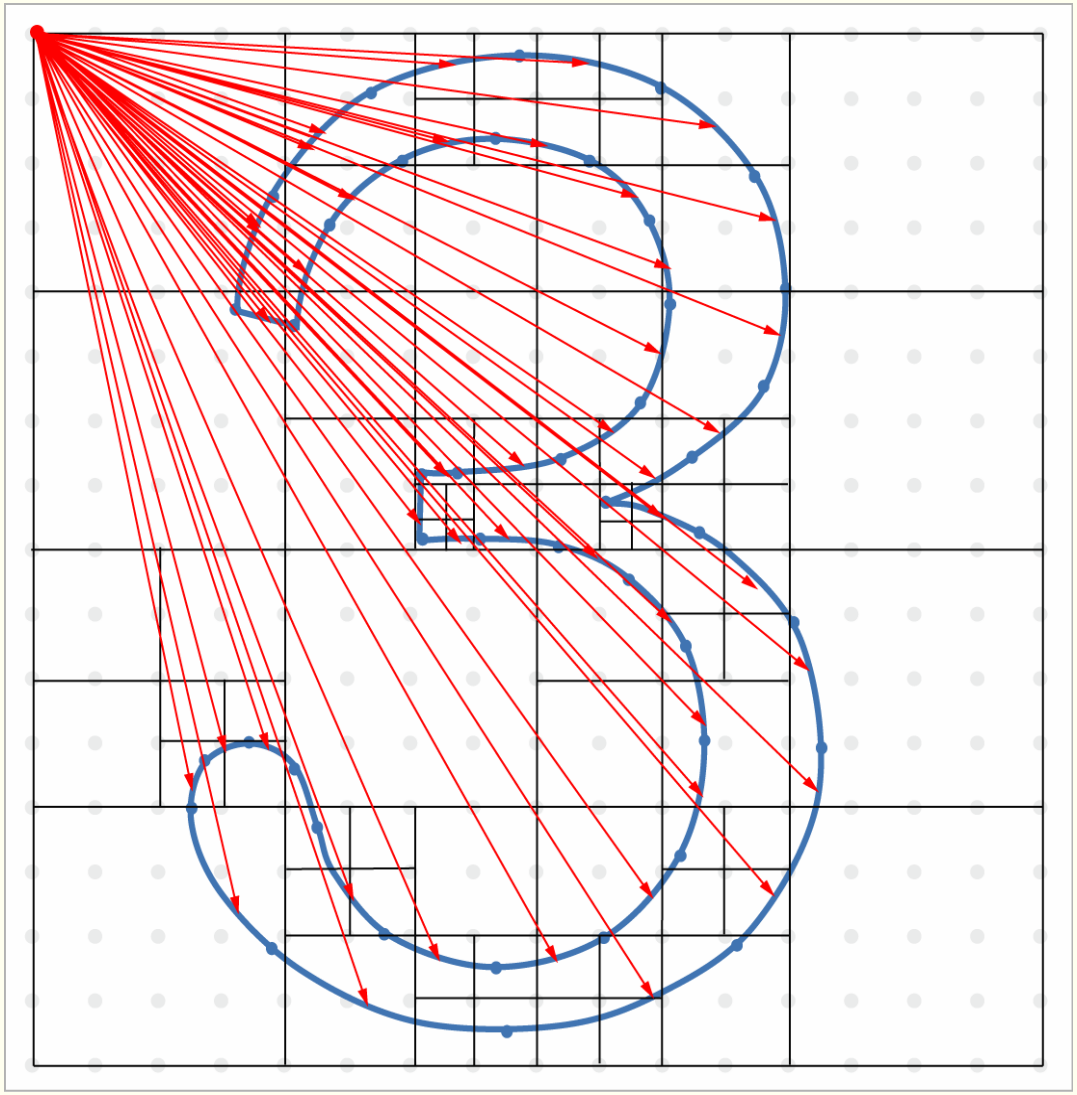
Outline Fonts

- Outline Fonts are defined by a series of straight line segments and curves
- We focus on fonts that use quadratic Bezier curves
  - Bezier curves are polynomial, parametric curves
  - Quadratic Bezier is a weighted average of three vertices or "control points"
  - A Bezier lies entirely within the convex hull of the control points



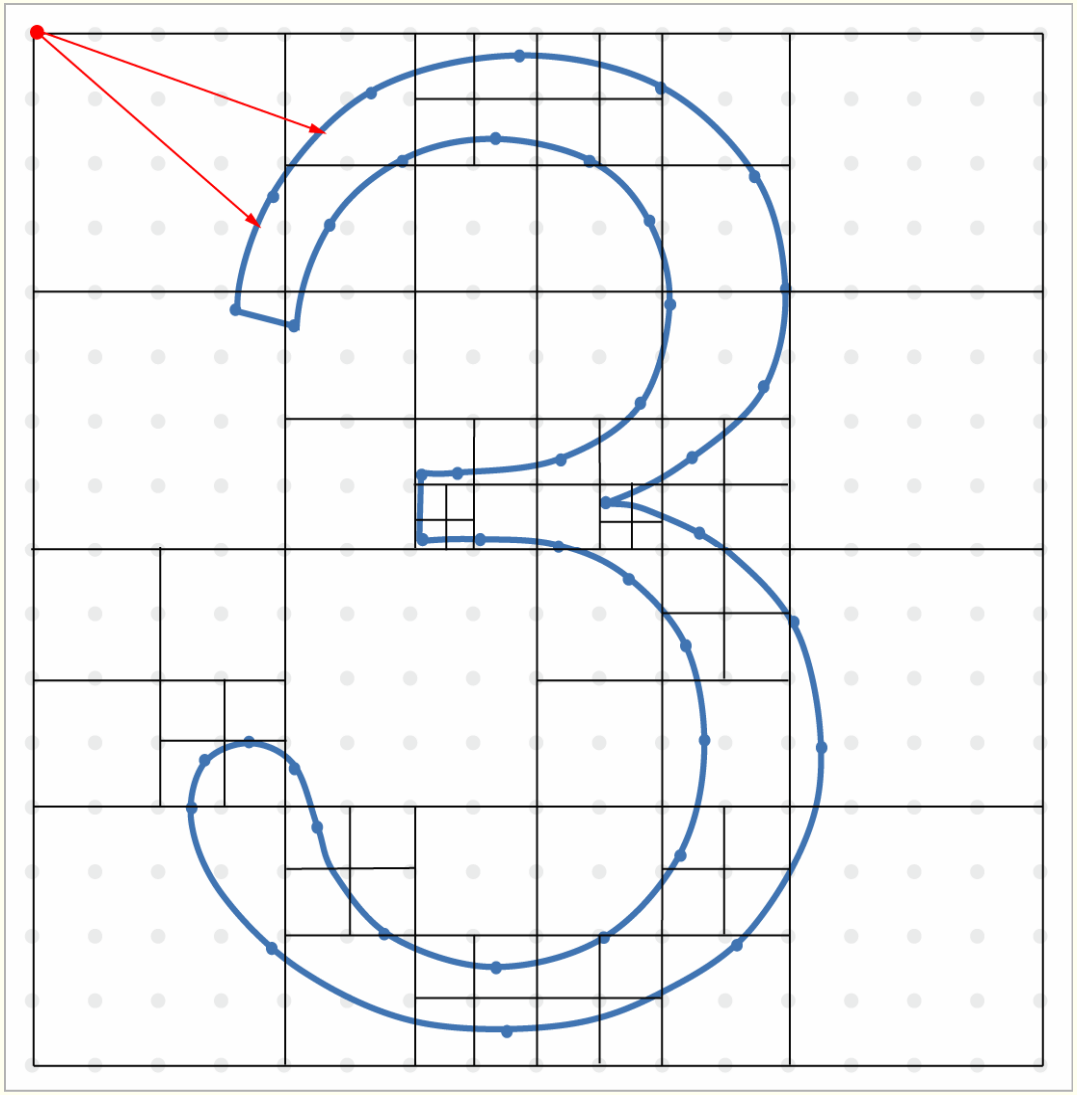
The Problem

- For each distance, we need to compute the distance to each Bezier curve in the glyph
- On average, 30 Bezier curves per glyph
  → Too much time!
- Needed a 10x speed up on the point-to-glyph distance computations



The Problem

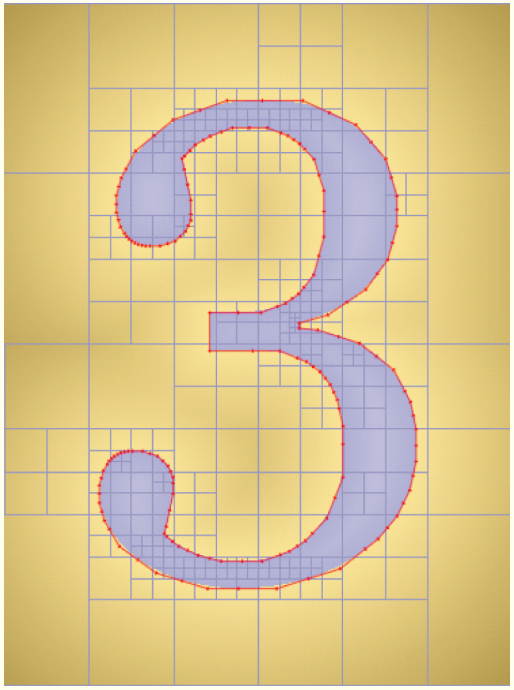
- It is unnecessary to compute the exact distance to each Bezier curve in the glyph
- Need a way to prune unnecessary distance computations

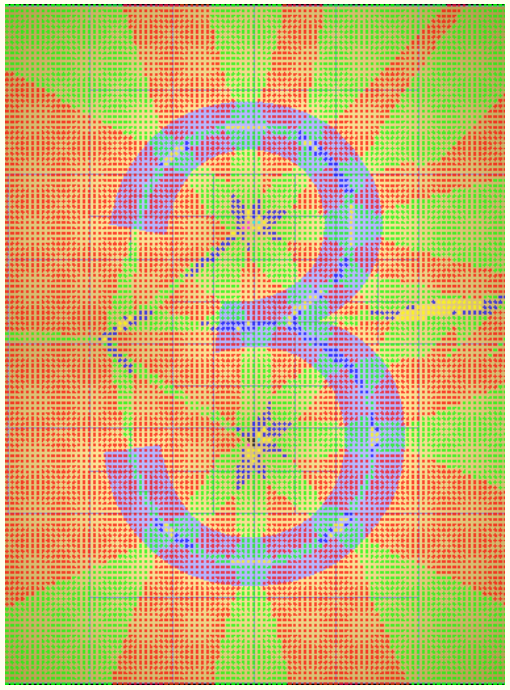


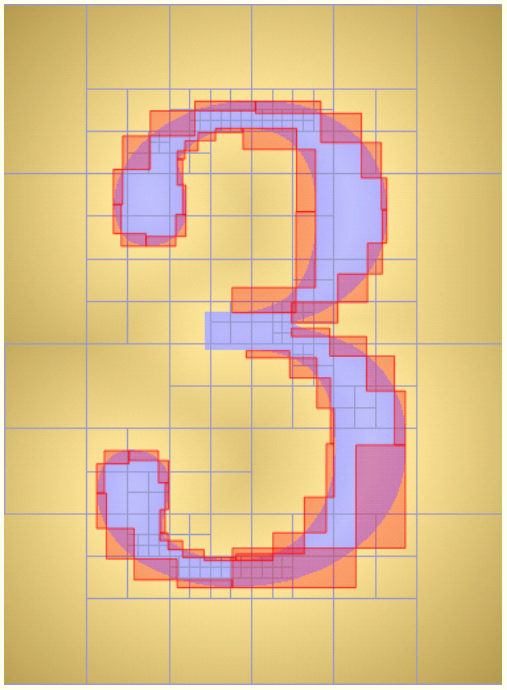
Possible Solutions

Possible Solutions

- Three general types of solutions
  - Curve approximation with line segments
  - Pruning using approximate distances from distance transforms
  - Pruning using bounding regions



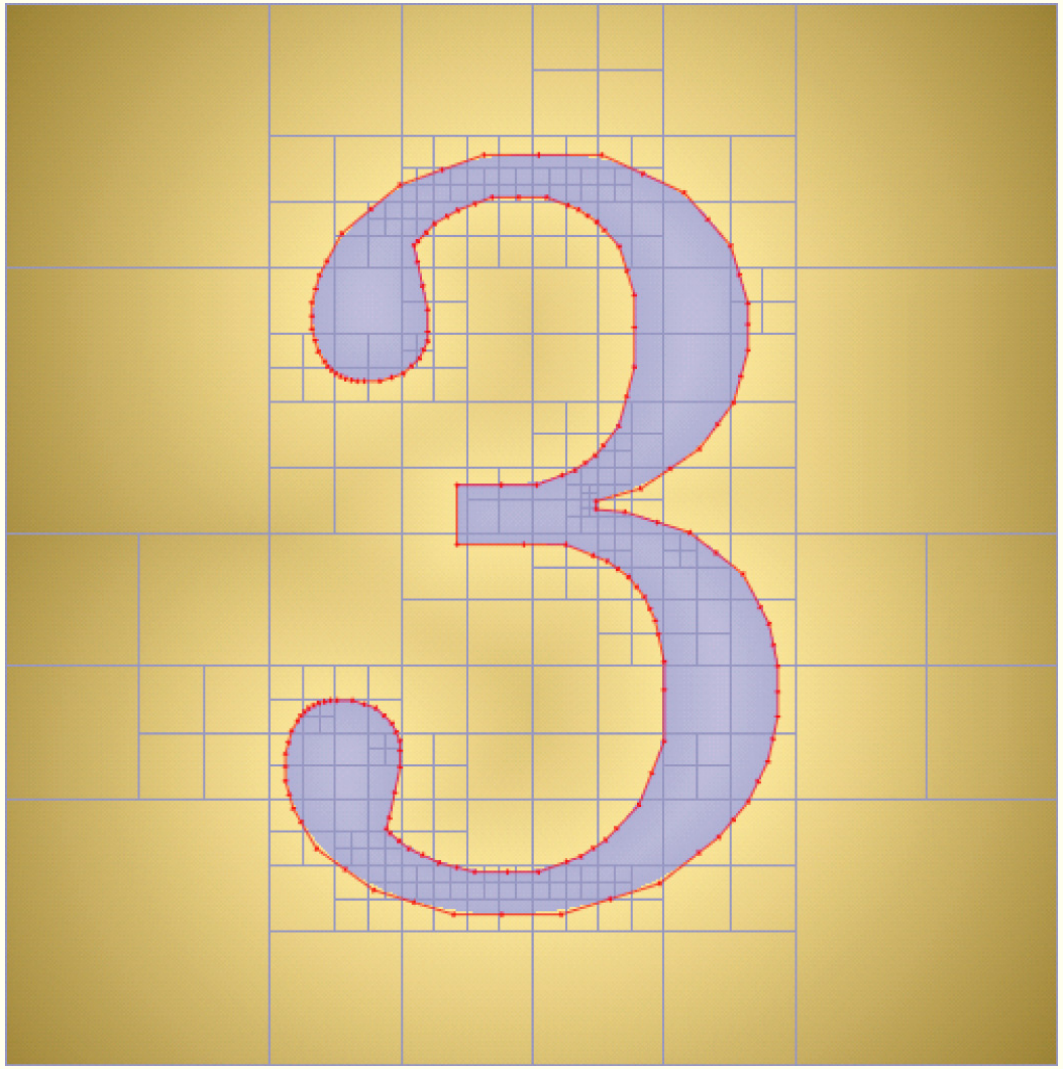




Line Segment Approximation

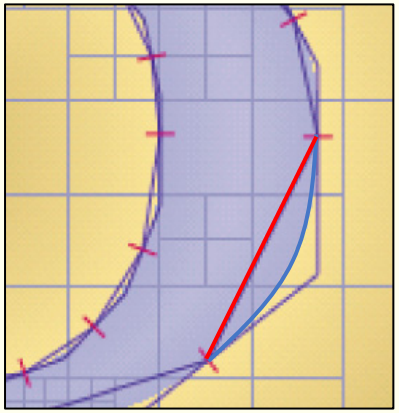
- Takes 1/20 the time to compute the distance from a point to a line segment than from a point to a quadratic Bezier curve

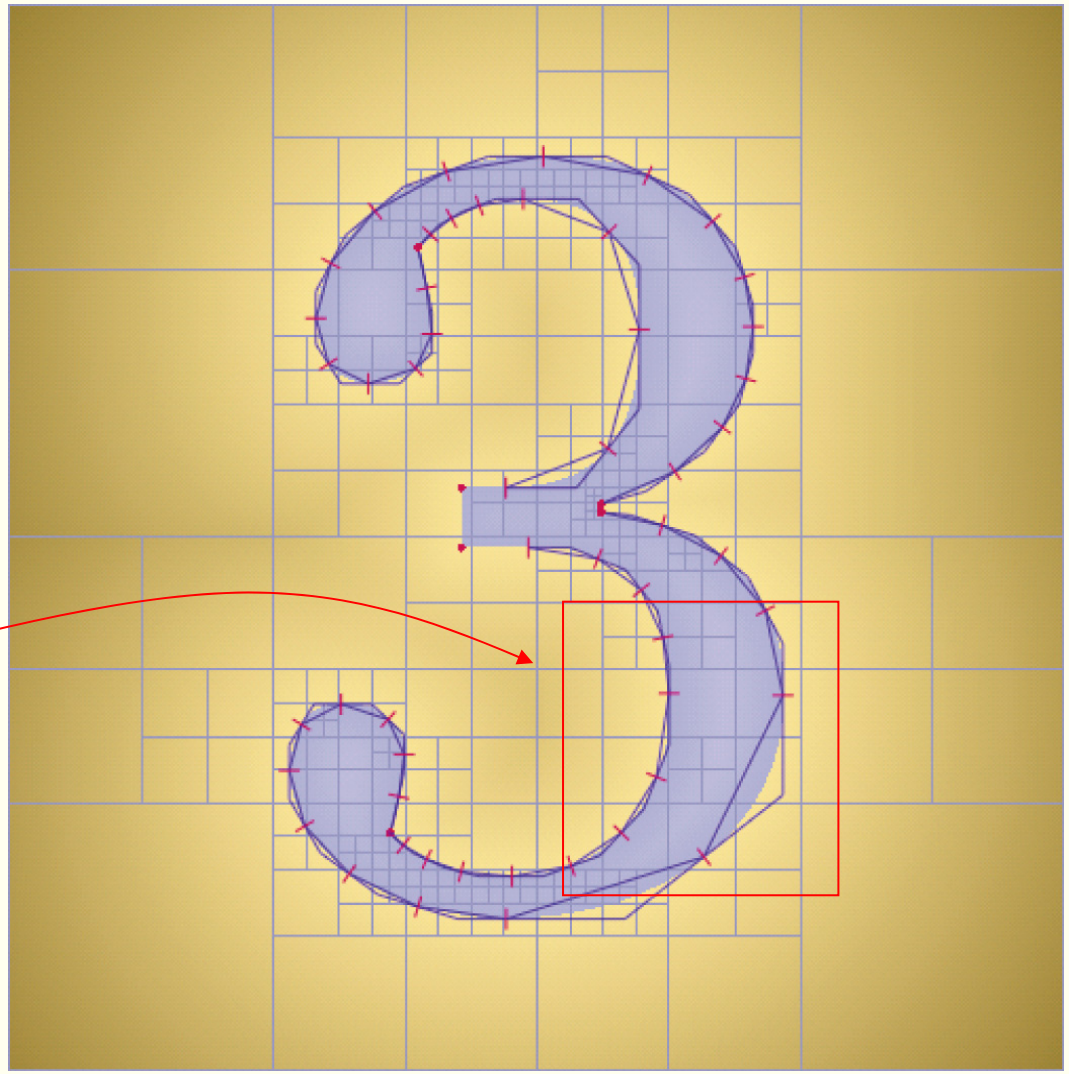
- Approximate quadratic Bezier curves with line segments



Line Segment Approximation

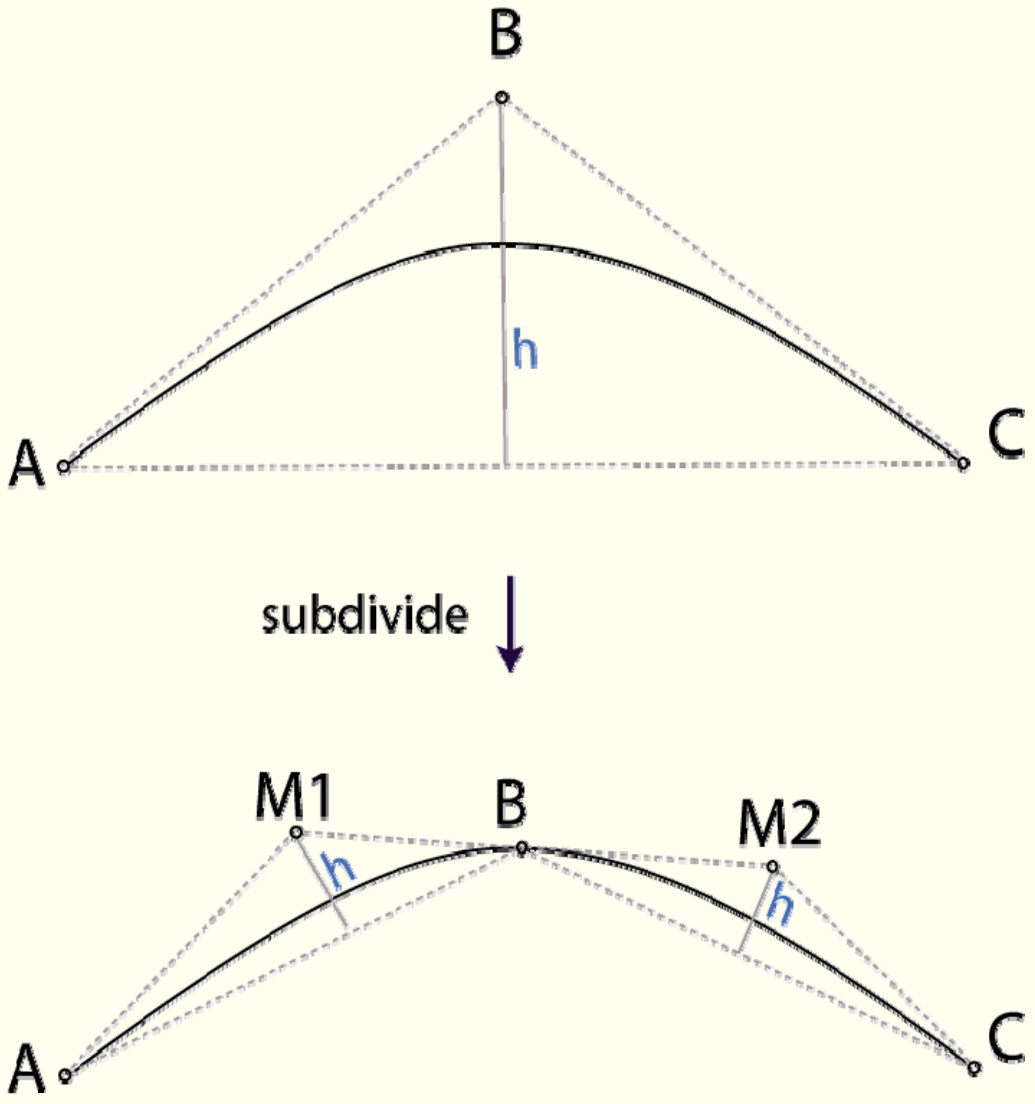
- Approximate a Bezier by the line segment joining its endpoints
- For a good approximation, Beziers need to be “sufficiently” flat





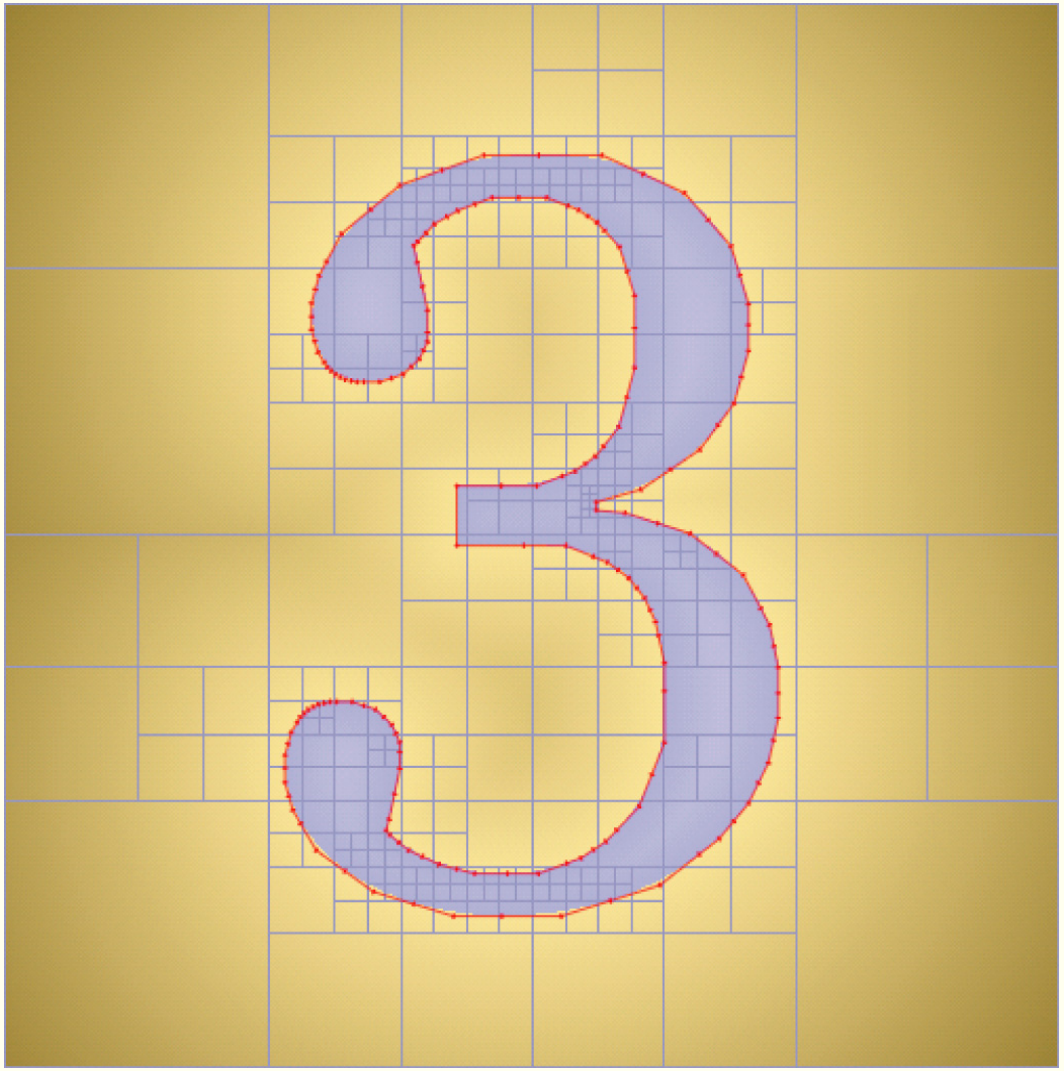
Line Segment Approximation

- A quadratic Bezier curve can be sub-divided in constant time into two parts
- This leads to "flatter" curves



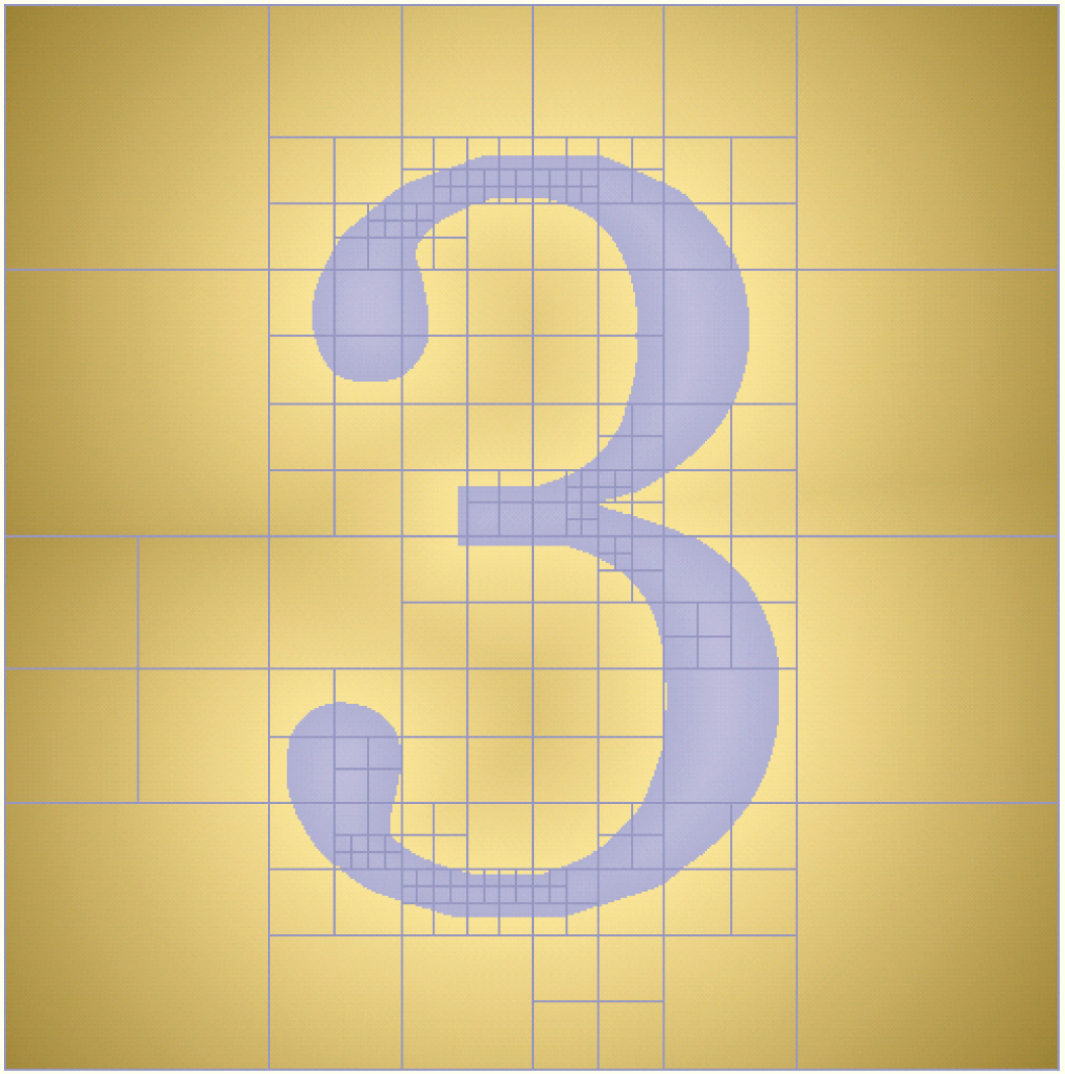
Line Segment Approximation

- For the Beziers to be sufficiently flat, we have to sub-divide too much



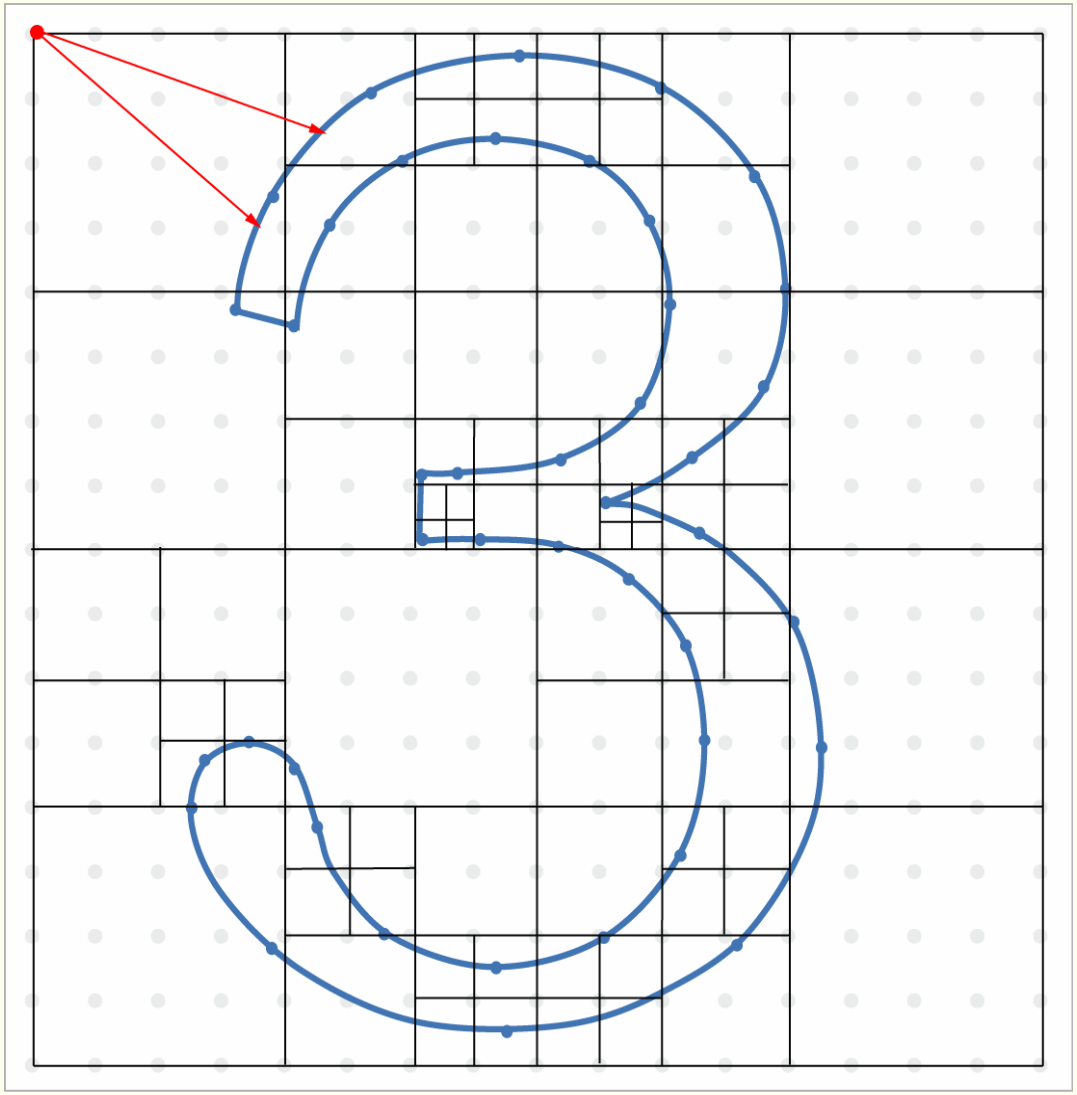
Line Segment Approximation

- For the Beziers to be sufficiently flat, we have to sub-divide too much
- And, we can still see corners



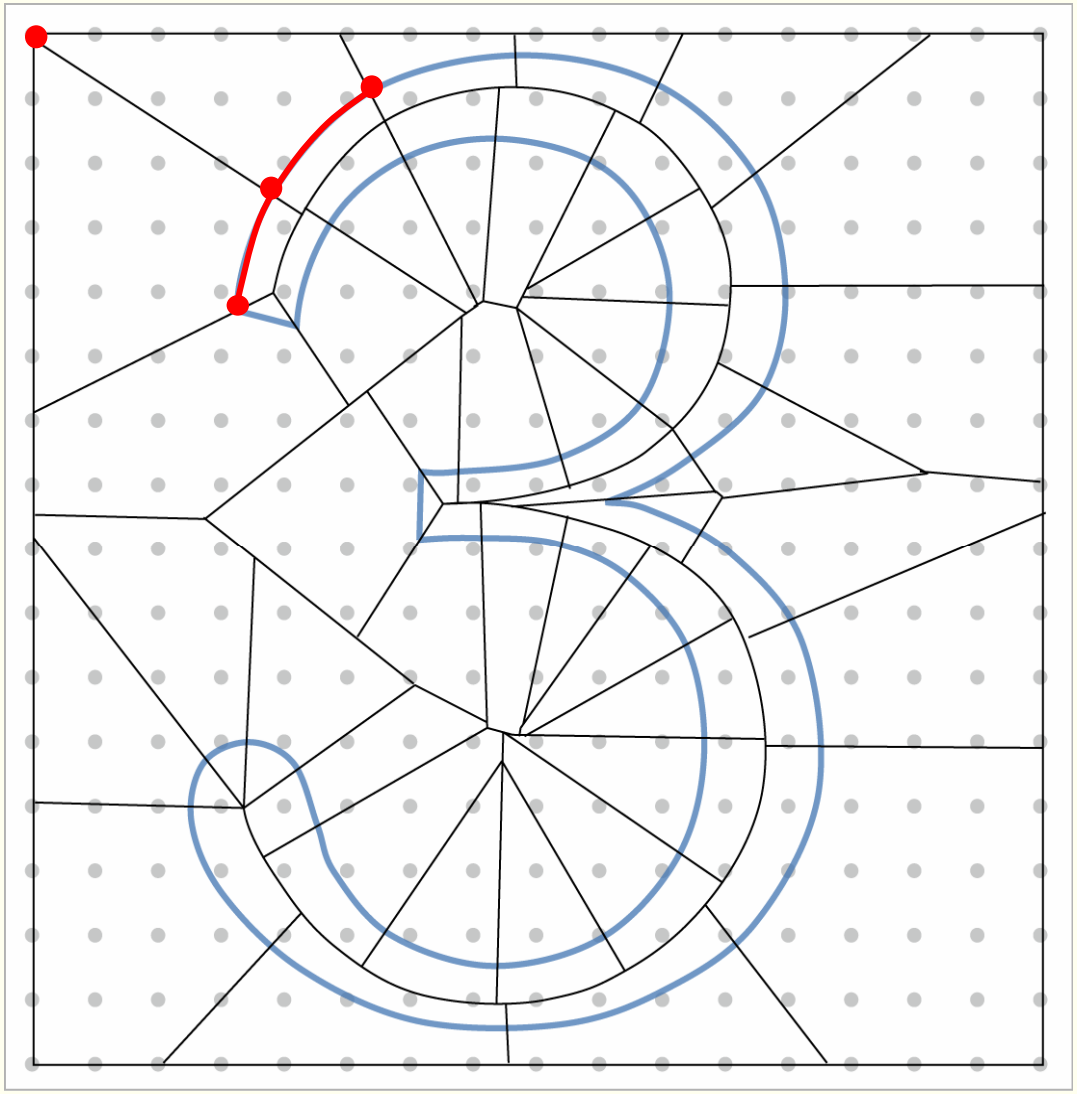
Pruning with Distance Transforms

- It is unnecessary to compute the exact distance to each Bezier curve in the glyph
- Need a way to prune unnecessary distance computations



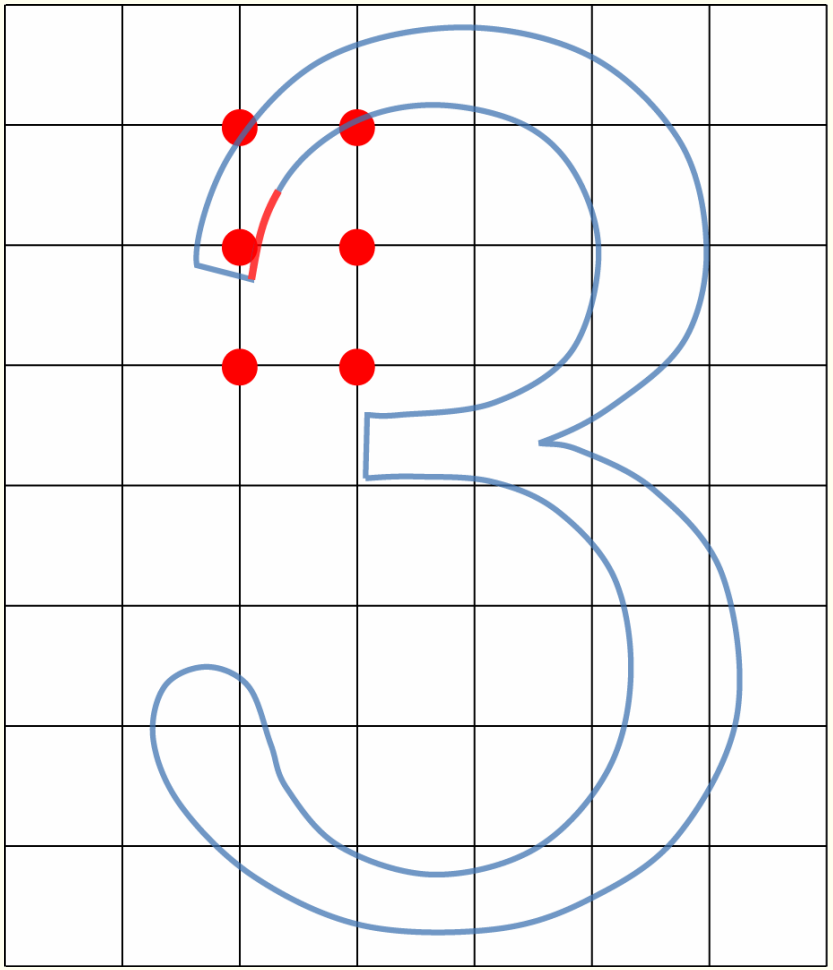
Pruning with Distance Transforms

- Suppose that for each grid point, we knew which Beziers were closest.
- A sort of discrete Voronoi diagram of the Bezier curves
- A **distance transform** uses an O(N) propagation method to **approximate** distances



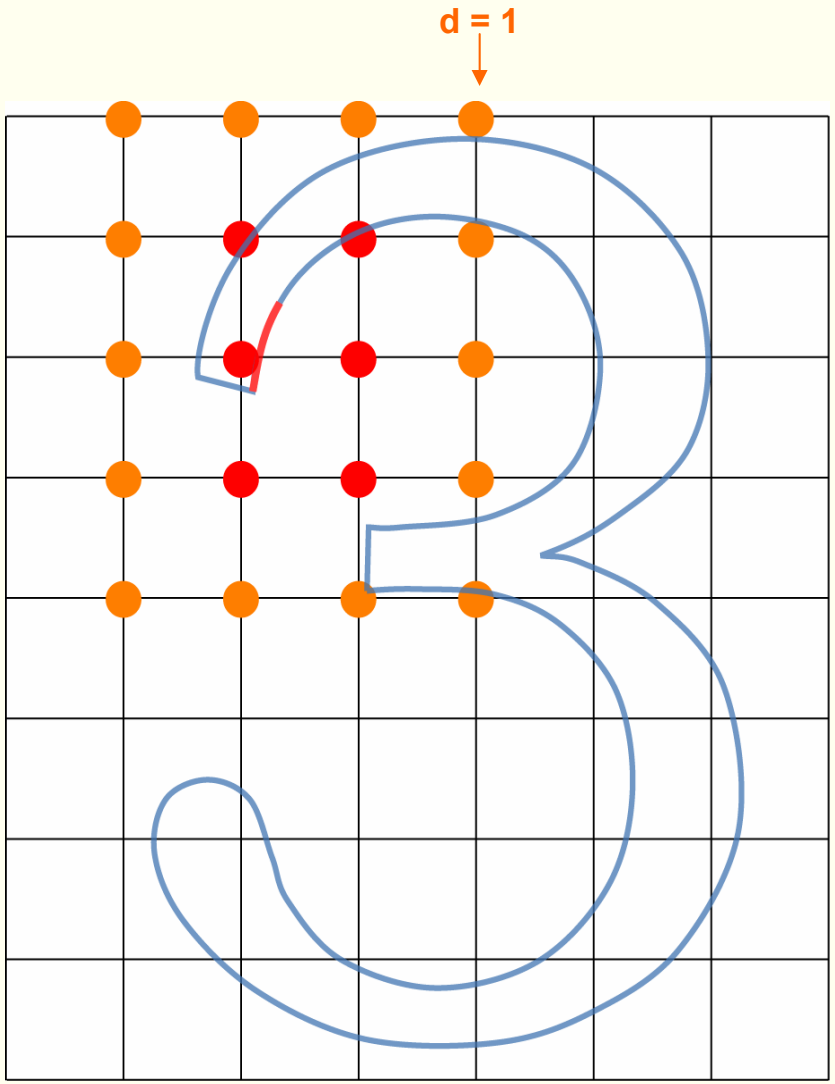
Pruning with Distance Transforms

- For each cell that contains a portion of the Bezier curve, set the distance of its corner vertices to d = 0



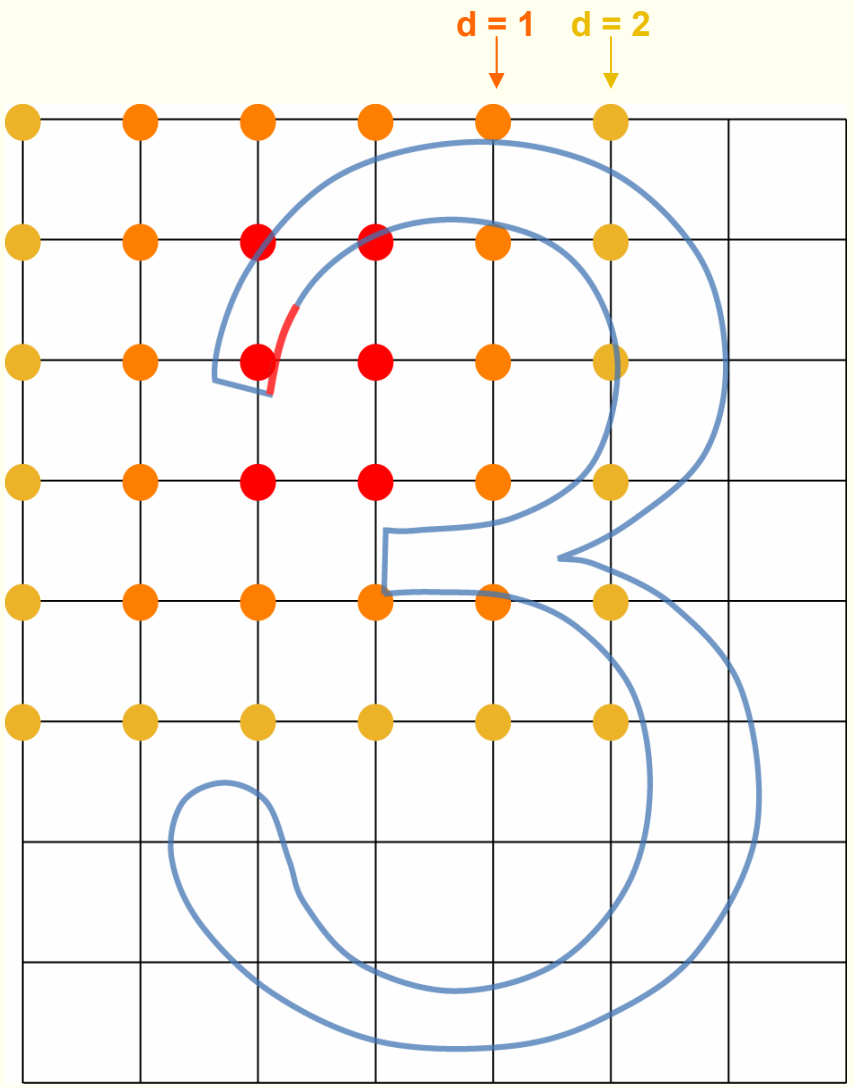
Pruning with Distance Transforms

- For each cell that contains a portion of the Bezier curve, set the distance of its corner vertices to d = 0
- Propagate the distance values outwards in flow-like manner



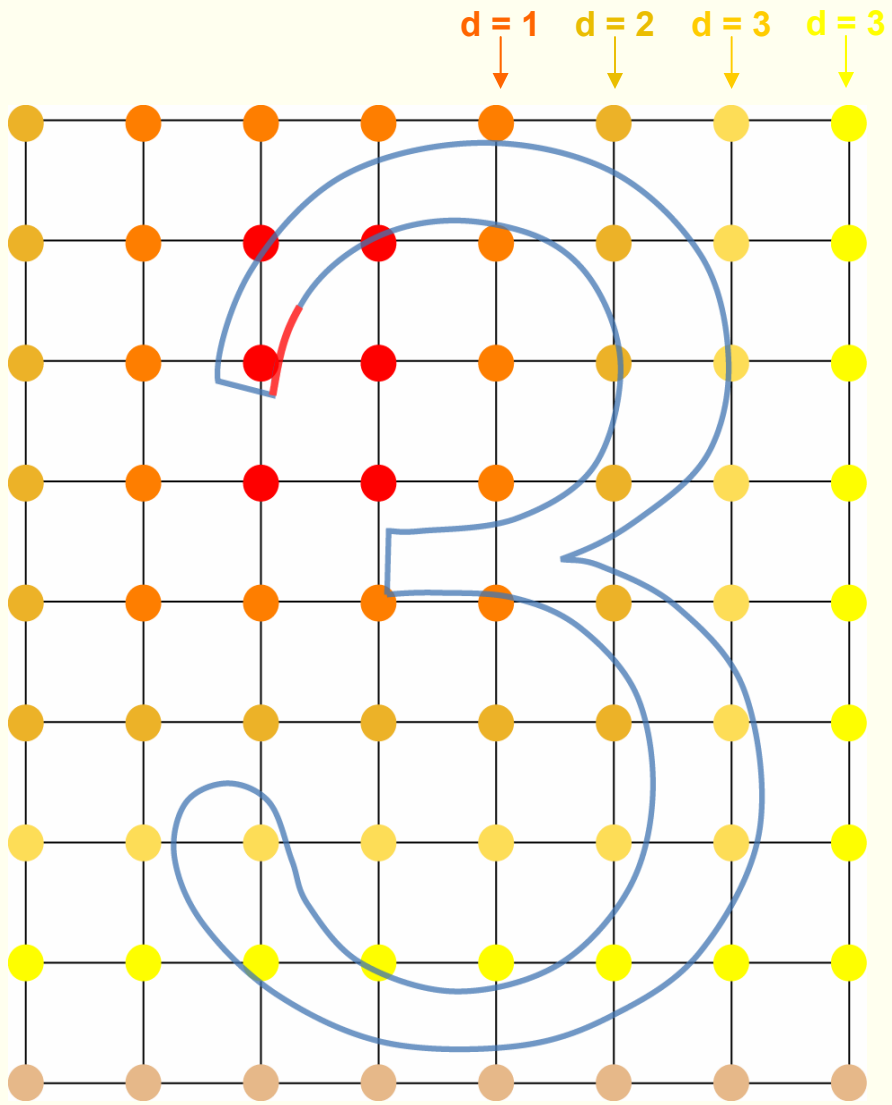
Pruning with Distance Transforms

- For each cell that contains a portion of the Bezier curve, set the distance of its corner vertices to d = 0
- Propagate the distance values outwards in flow-like manner



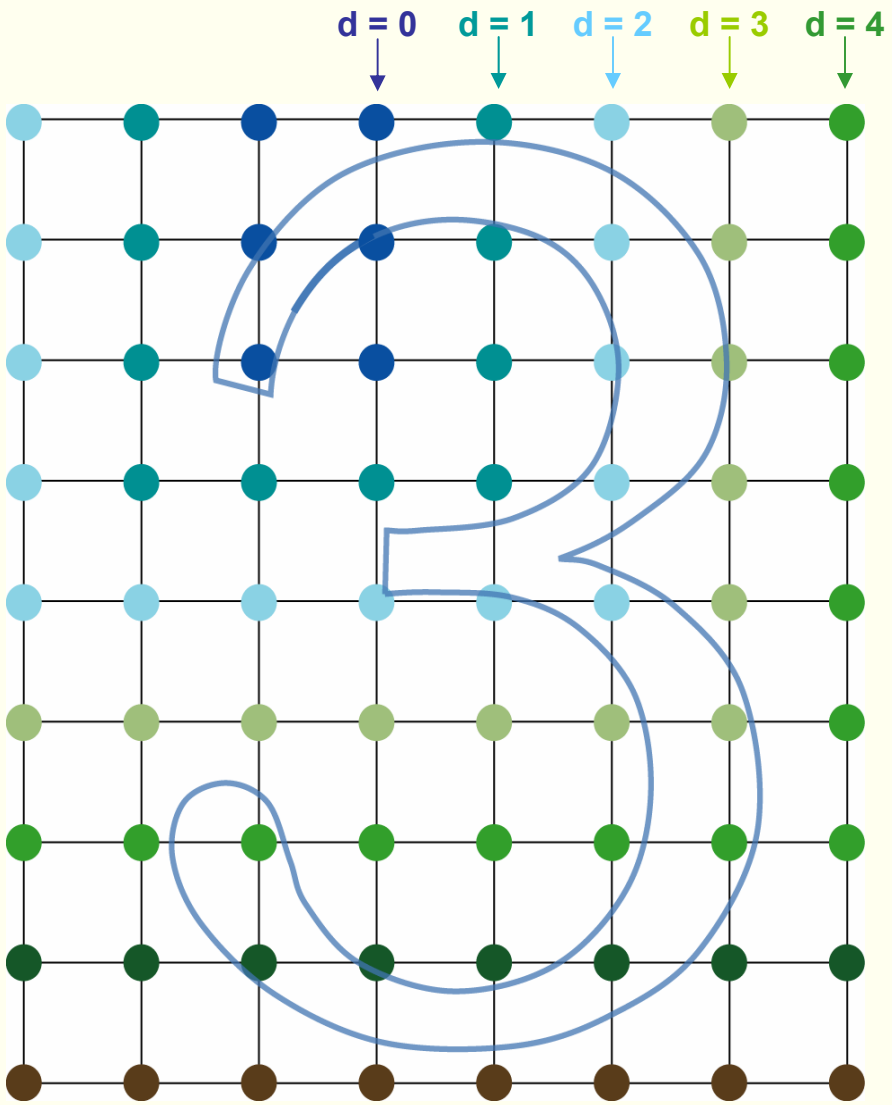
Pruning with Distance Transforms

- For each cell that contains a portion of the Bezier curve, set the distance of its corner vertices to d = 0
- Propagate the distance values outwards in flow-like manner



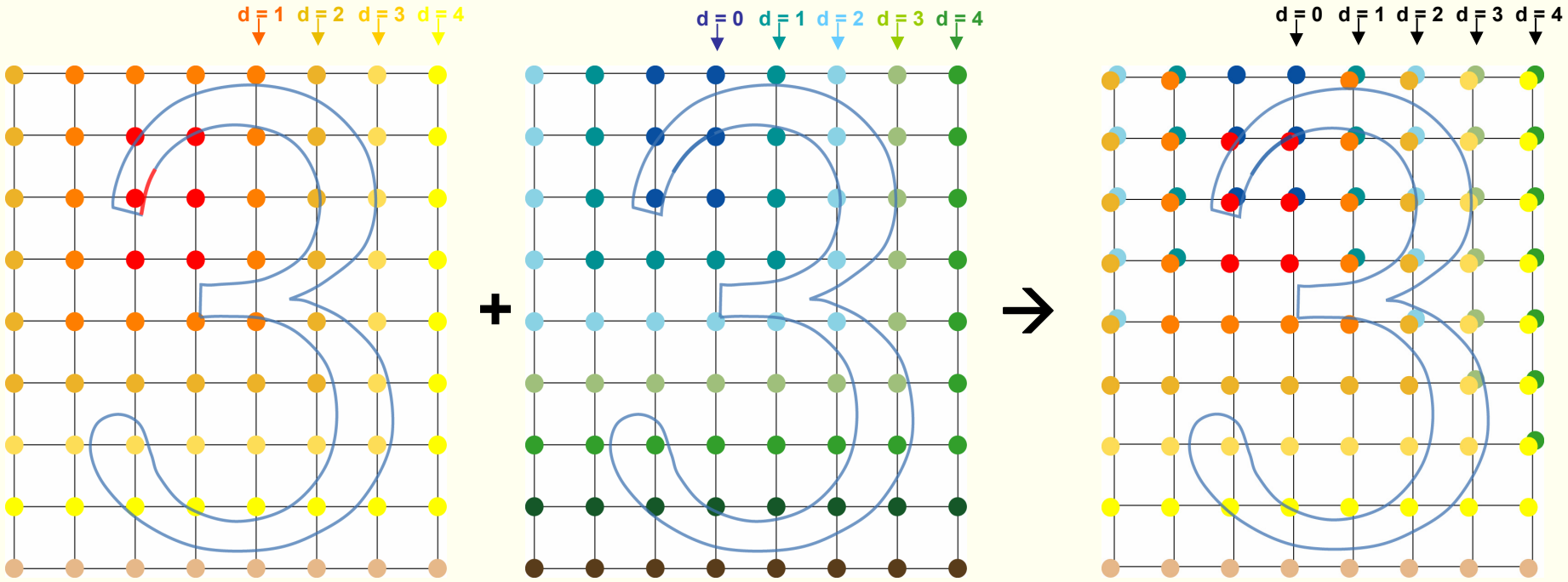
Pruning with Distance Transforms

- Repeat for each Bezier curve in the outline



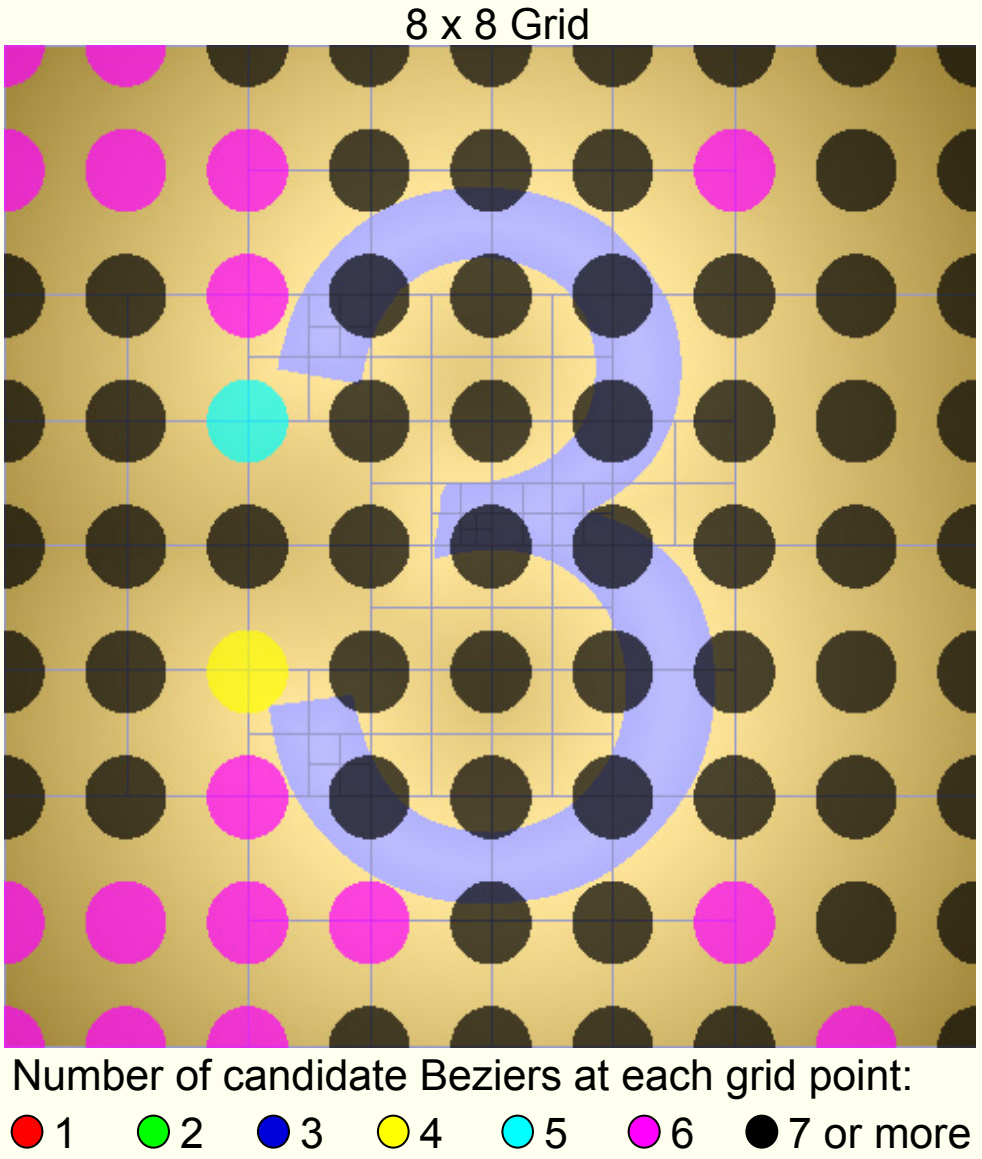
Pruning with Distance Transforms

- For each grid points, tally which Beziers can be the closest
- How many candidate Beziers can a grid point have?
  → It depends on the type of distance transform & grid size



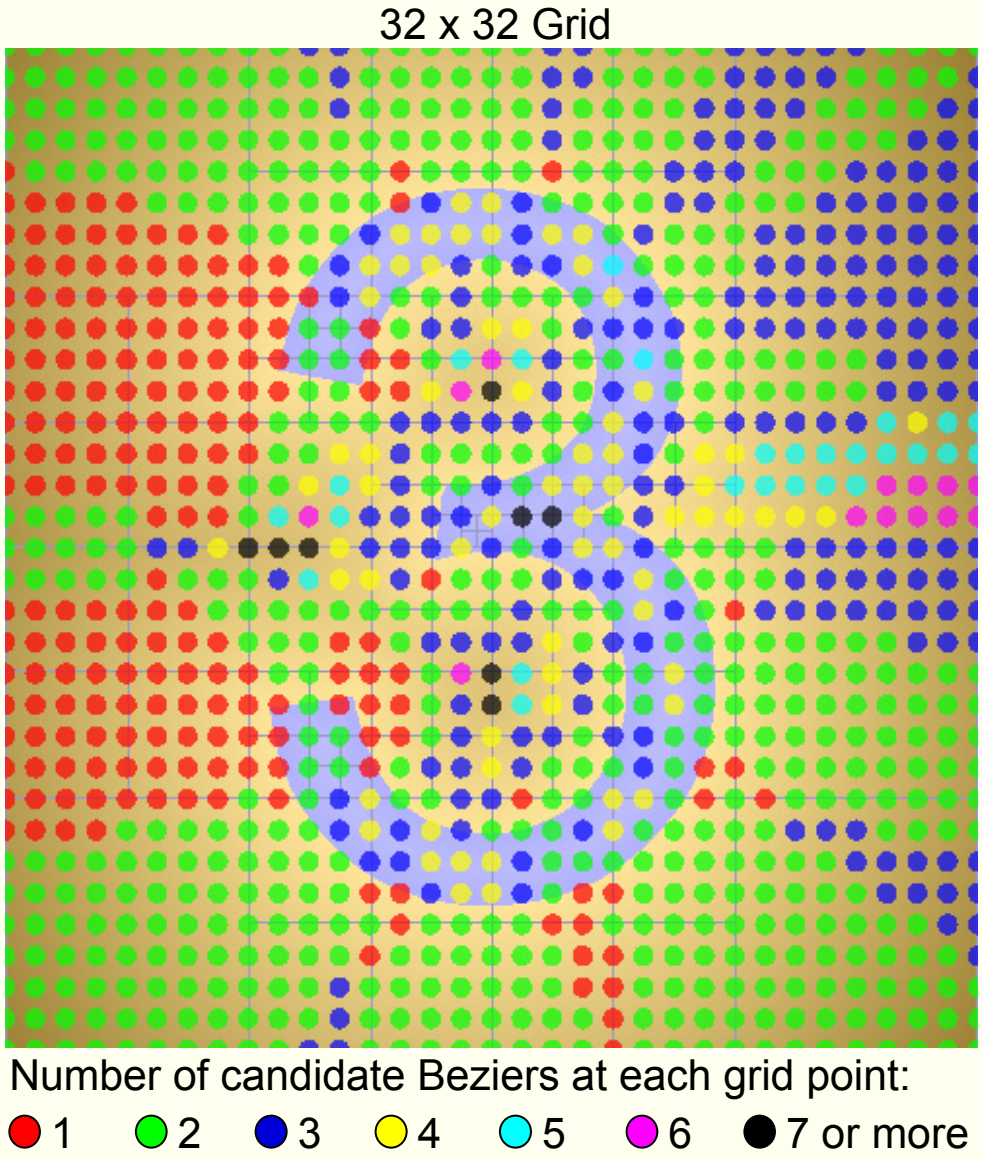
Pruning with Distance Transforms

- We would like to have only 1 or 2 candidate Beziers per grid point
- How fine a grid does this require?



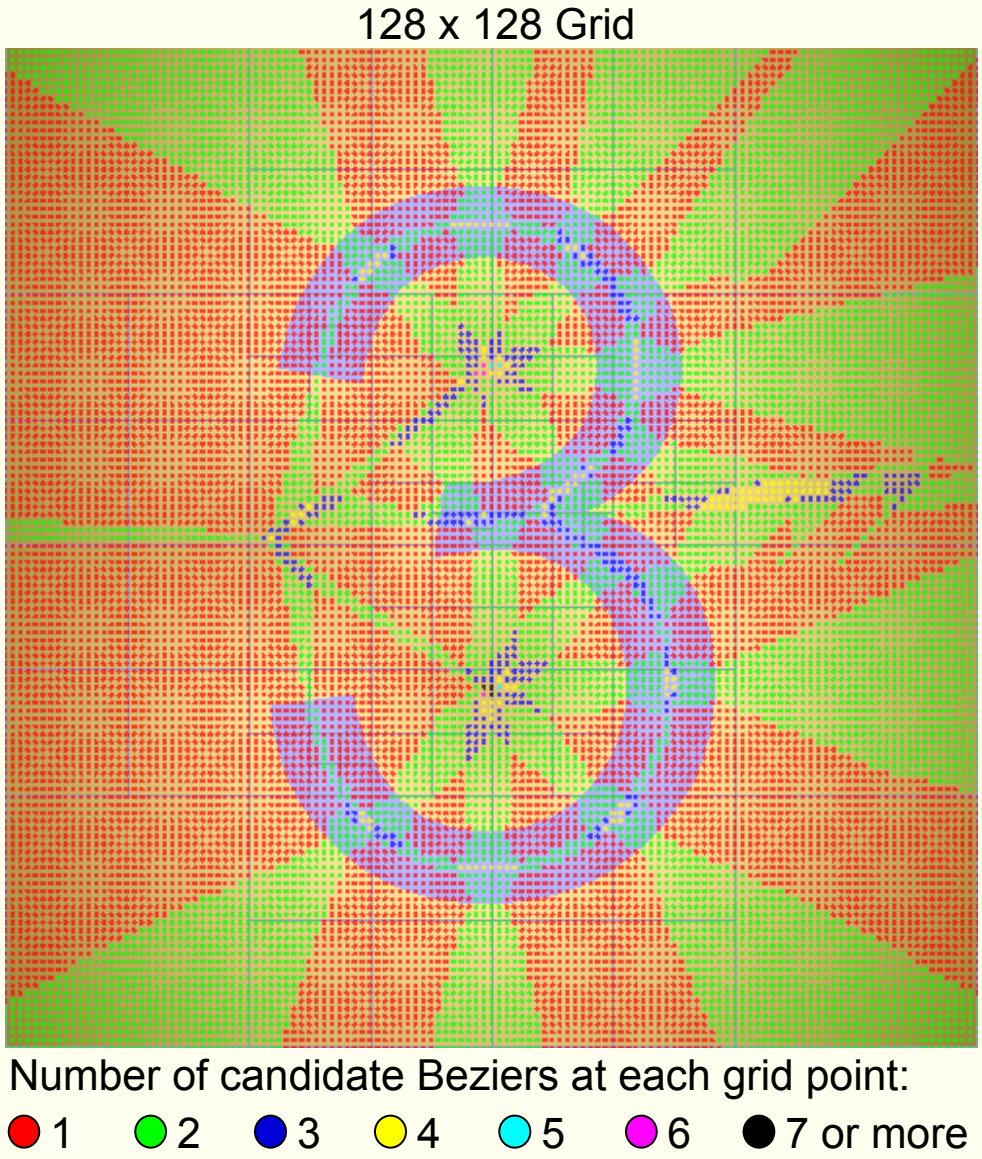
Pruning with Distance Transforms

- We would like to have only 1 or 2 candidate Beziers per grid point
- How fine a grid does this require?



Pruning with Distance Transforms

- We would like to have only 1 or 2 candidate Beziers per grid point
- How fine a grid does this require?
  - → either 64 x 64 and 128 x 128 would probably suffice but it takes too long to propagate the distances

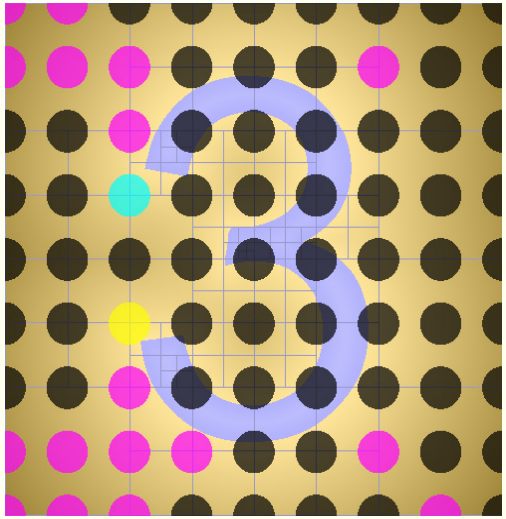


Pruning with Distance Transforms

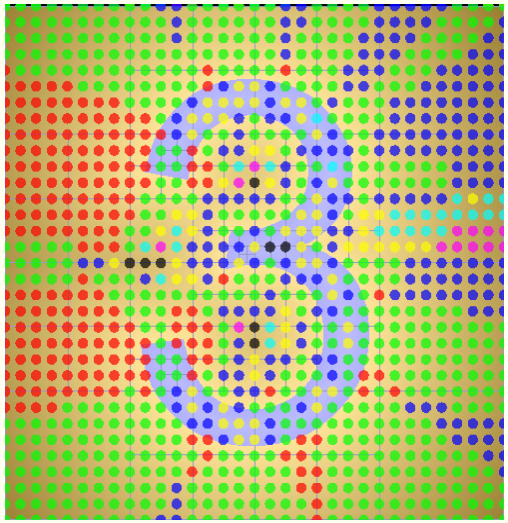
Method (Arial Font)	Glpyhs / Sec	Conics / Distance call
Naive	107.91	13.92
Distance Transform*: 8 x 8	298.55	9.04
Distance Transform*: 32 x 32	474.53	2.22
Distance Transform*: 128 x 128	140.78	1.30

* Times from using the Eight Signed Sequential Euclidean Distance Transform

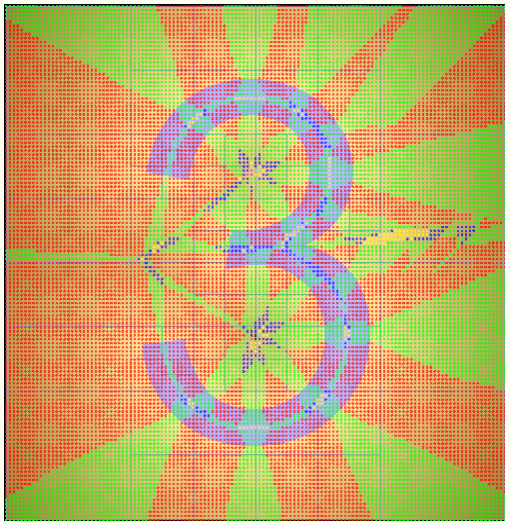
8 x 8 Grid



32 x 32 Grid



128 x 128 Grid

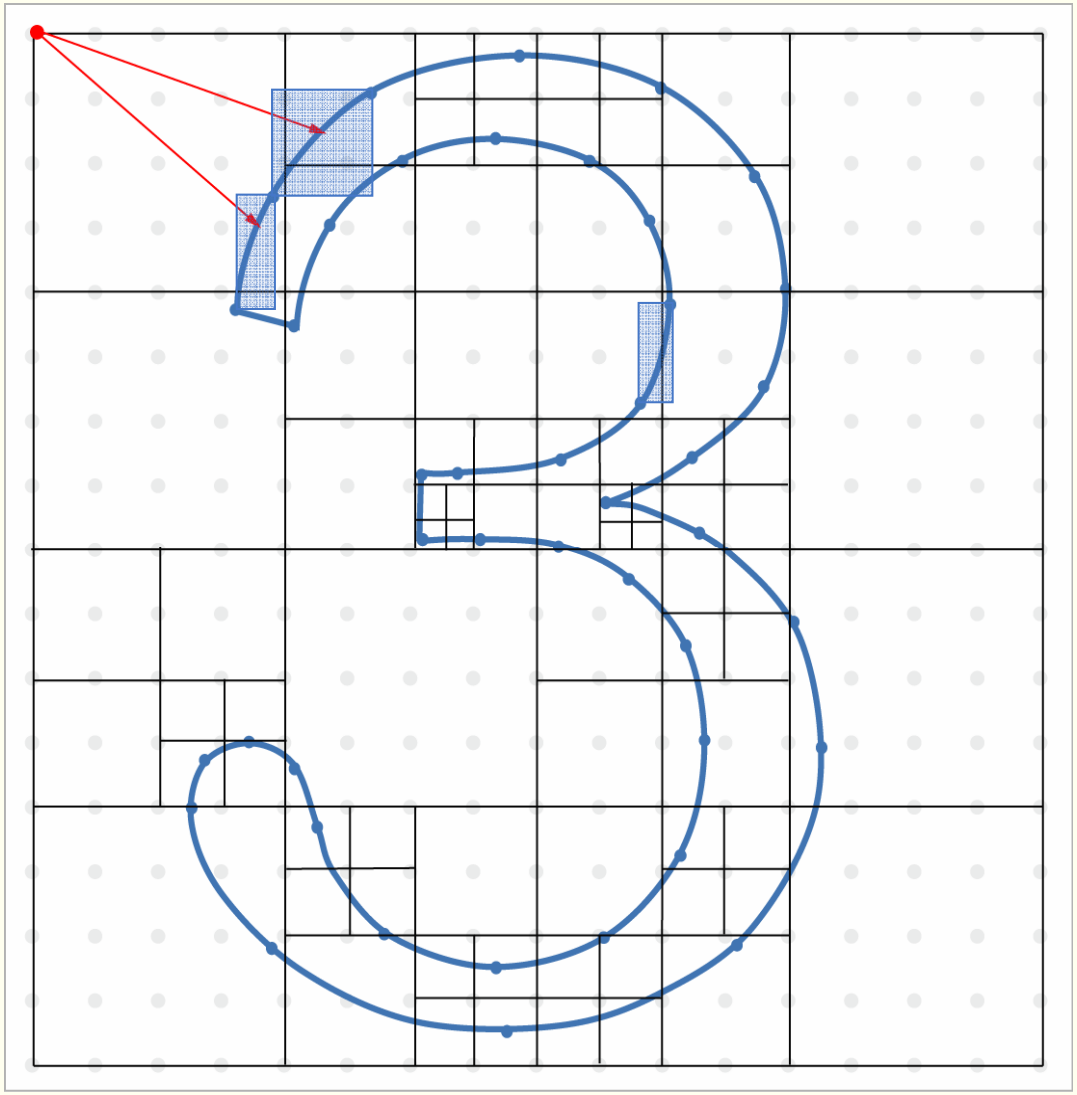


Number of candidate Beziers at each grid point: 1 2 3 4 5 6 7 or more

Elena Jakubiak. Tufts University, Department of Computer Science

Pruning with Bounding Regions

- Compute the distance to each bounding region
- Quicker than computing the distance to the Bezier curve
- Prune Bezier curves

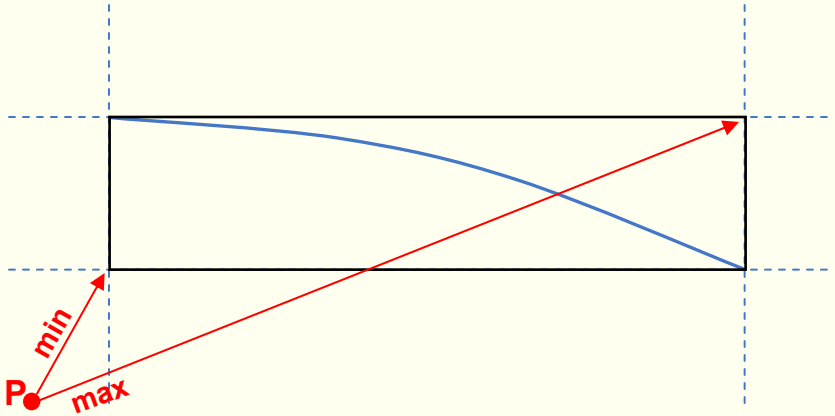


Pruning with Bounding Regions

- Consider the axis aligned bounding box of a quadratic Bezier curve

- Quickly compute the minimum and maximum distance from P to the bounding box

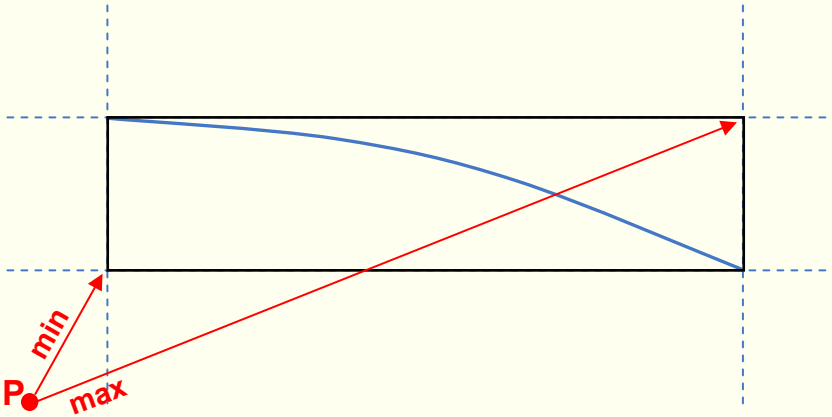
- Provides bounds on the distance from P to the Bezier curve

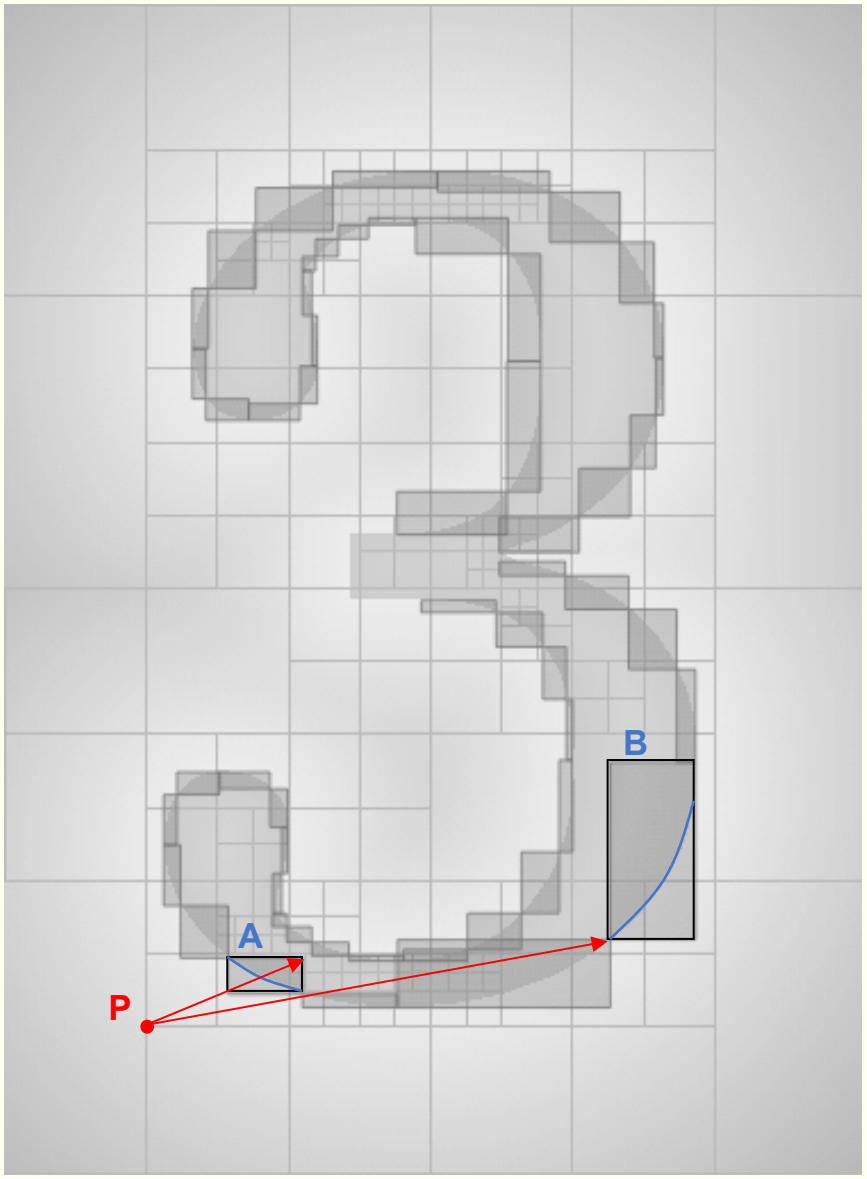


Pruning with Bounding Regions

- Consider a given query point P and two Bezier curves A and B along with their bounding boxes

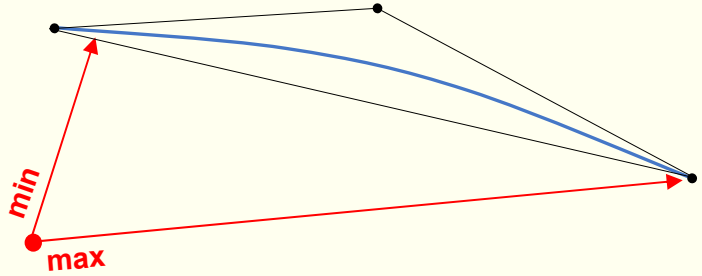
- The maximum distance to A is less than the minimum distance to B → we can eliminate B

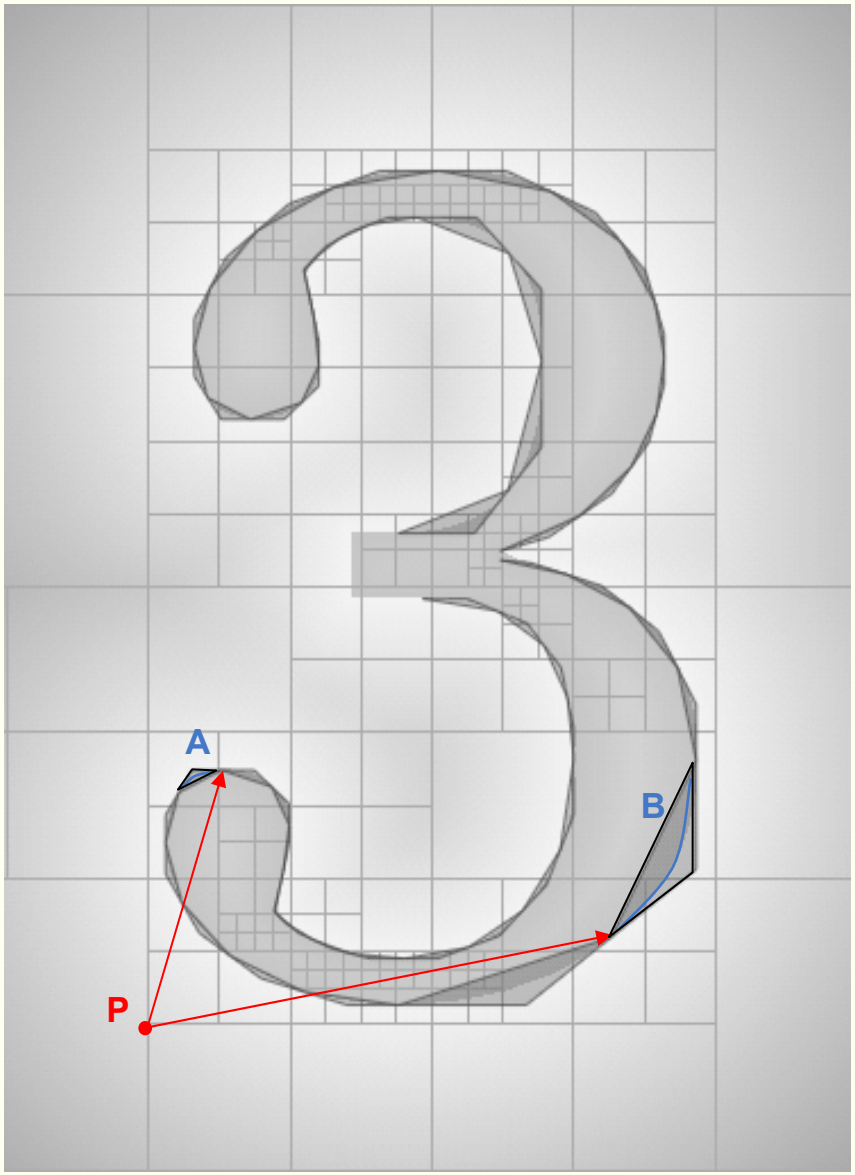




Pruning with Bounding Regions

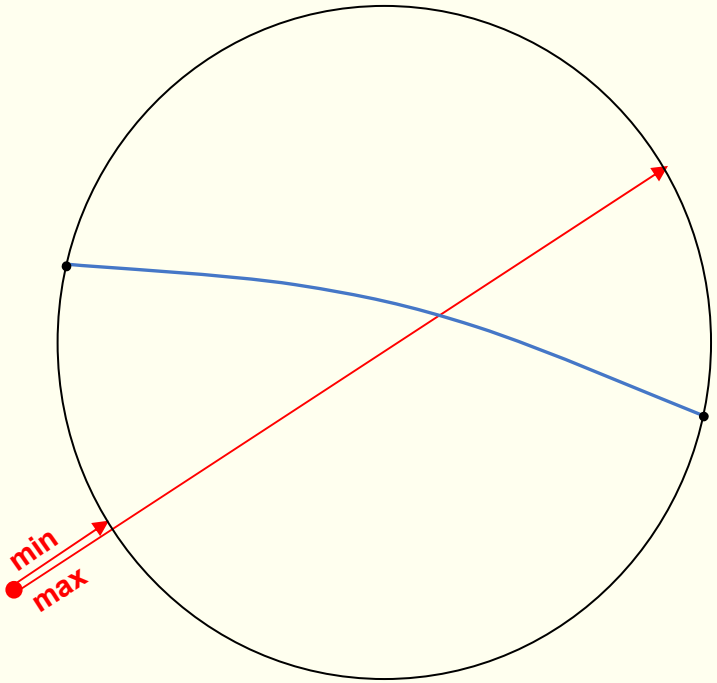
- Or, prune Beziers using bounding triangles –the convex hull of the control vertices

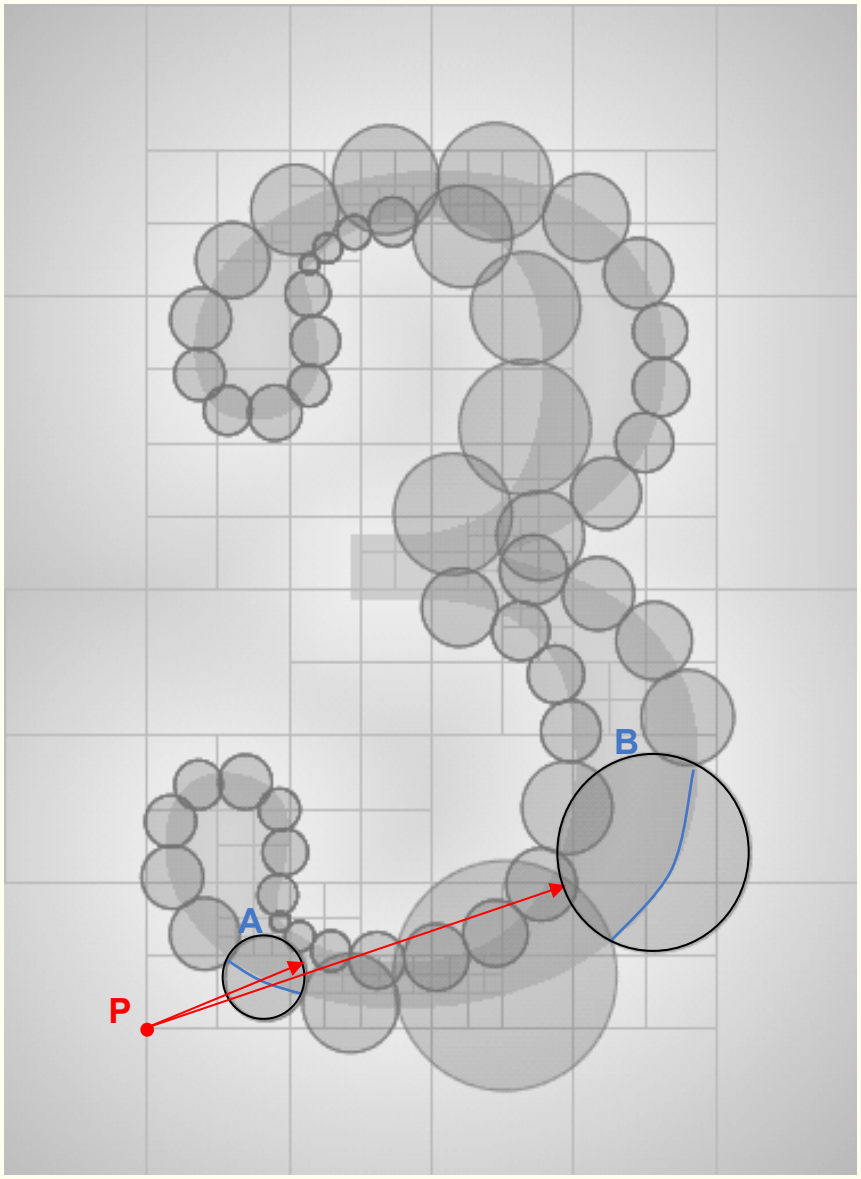




Pruning with Bounding Regions

- Or, prune Beziers using bounding circles



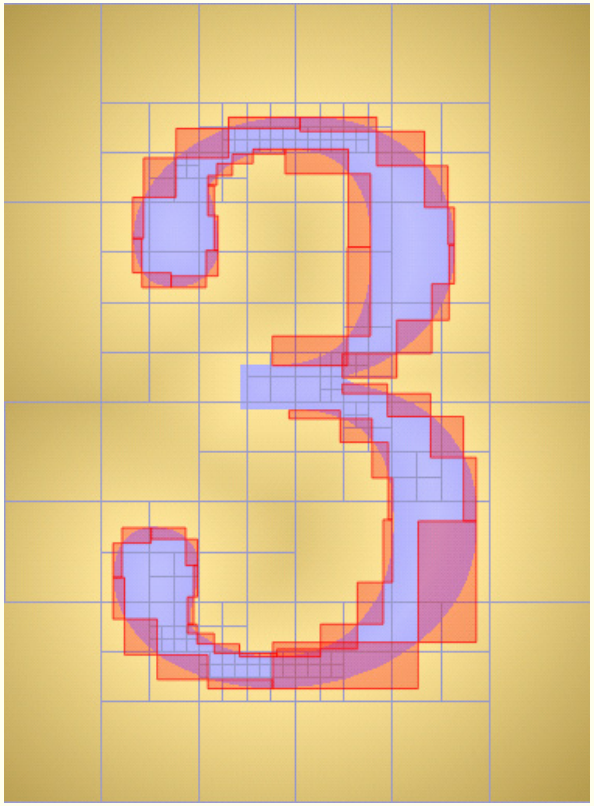


Pruning with Bounding Regions

Axis Aligned Bounding Boxes

Pro: Quick point-to-bounding region distance computation

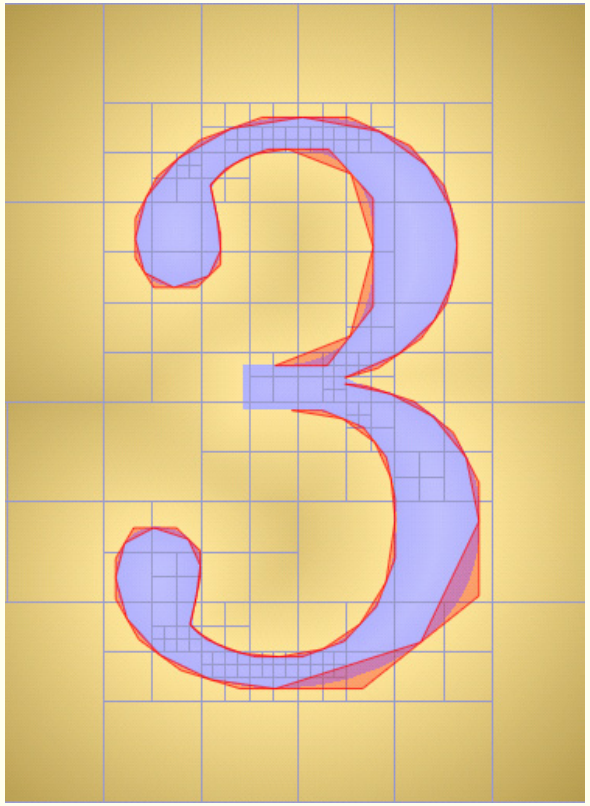
Con: Excessive area coverage for diagonal curves



Control Vertices Convex Hull

Pro: Very tight area coverage

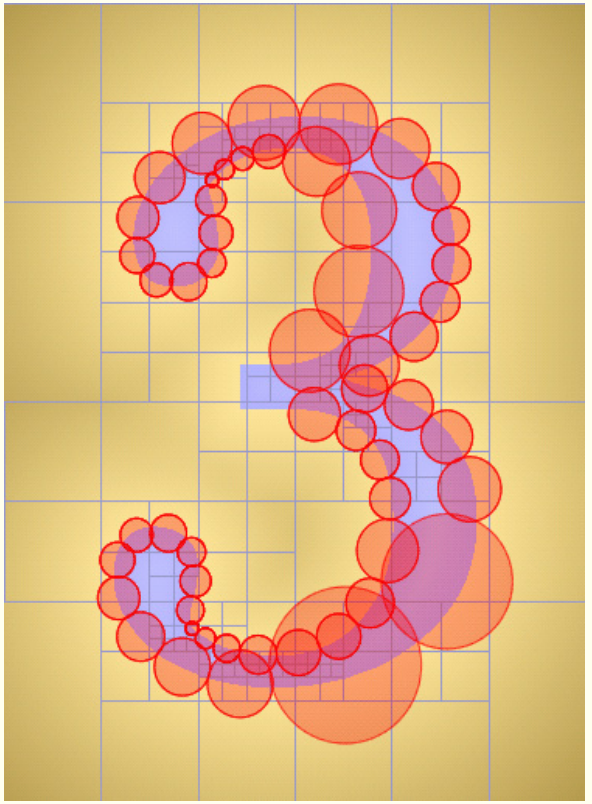
Con: Most expensive point-to-bounding region distance computation



Control Vertices Convex Hull

Pro: Very fast point to bounding region computation

Con: Excessive area coverage overall

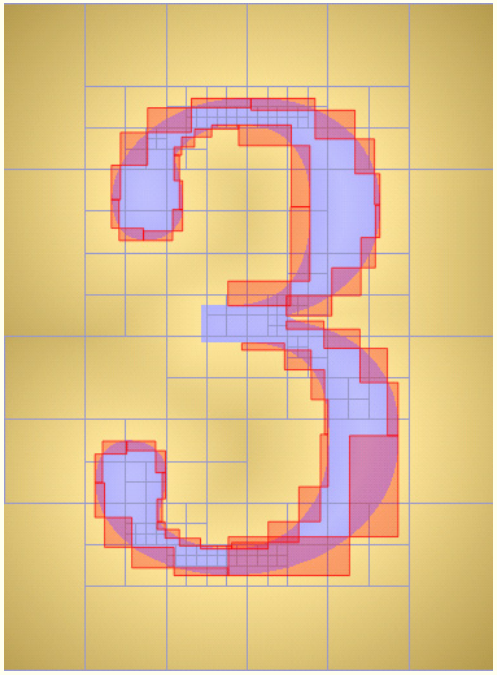


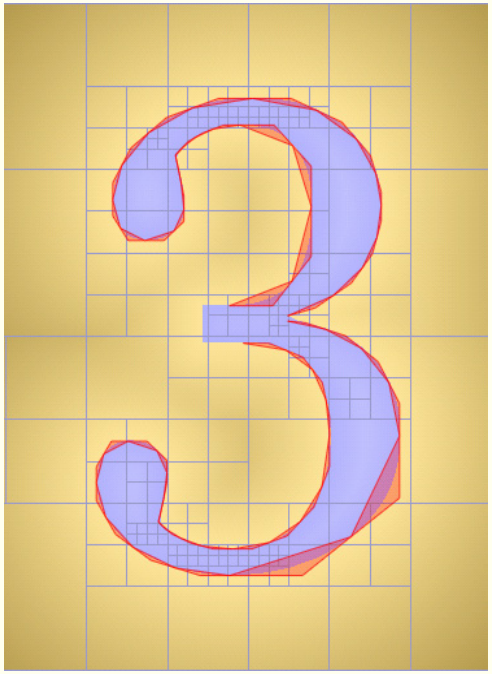
Pruning with Bounding Regions ≈ 5x

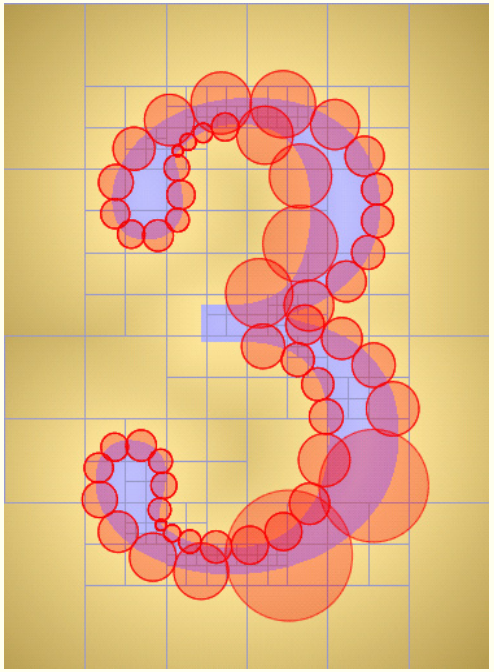
Font	Method	Glpyhs / Sec	Conics / Distance call
Arial	Naïve	107.91	13.92
	Bounding Boxes	583.54	1.13
	Bounding Triangles*	353.66	0.88
	Bounding Circles	534.31	1.41
Georgia	Naïve	37.67	28.65
	Bounding Boxes	279.08	1.41
	Bounding Triangles*	168.01	1.08
	Bounding Circles	271.74	1.74

***Approximation based on relative timing from another computer**

Method	< > =	* /	√	+ -
Bounding Boxes	4	4	0	6
Bounding Triangles	6	20	1	15
Bounding Circles	1	4	0	4





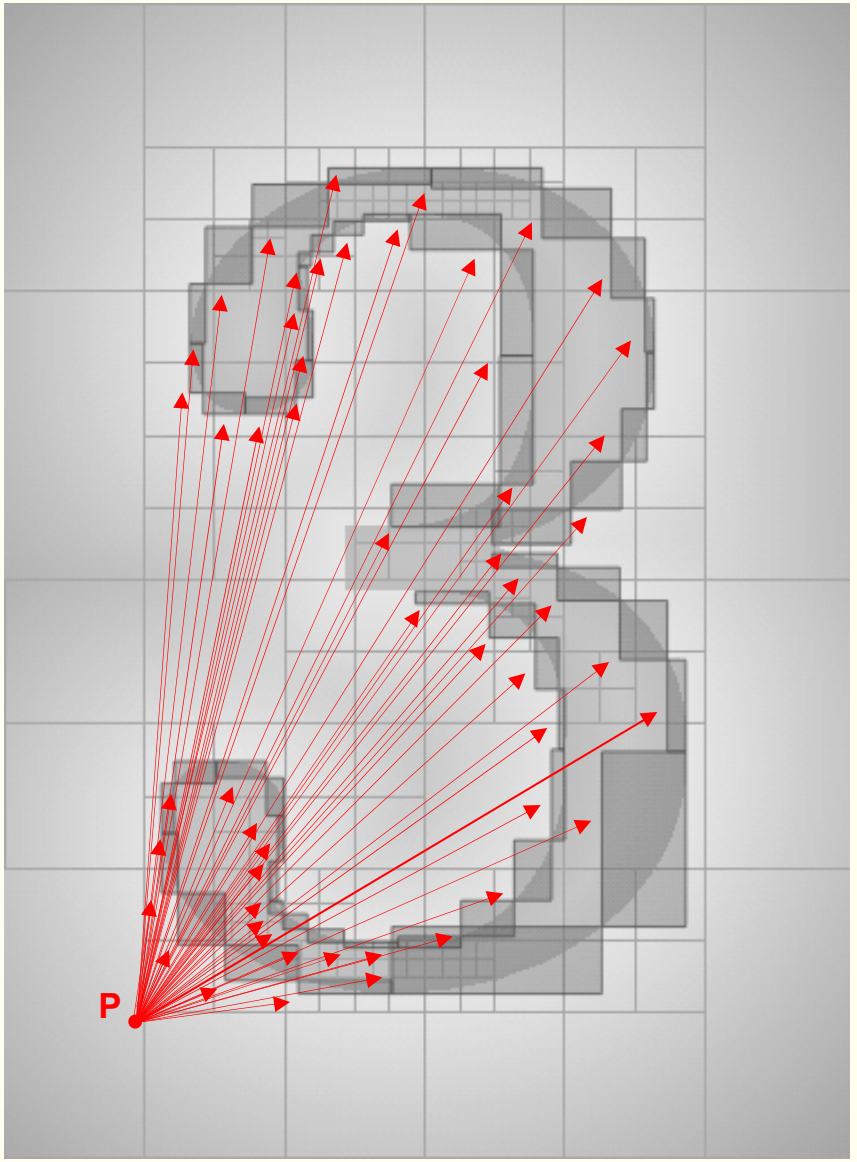


Elena Jakubiak. Tufts University, Department of Computer Science

Pruning with Bounding Regions

- Still computing the distance from **each** query point to **each** bounding box

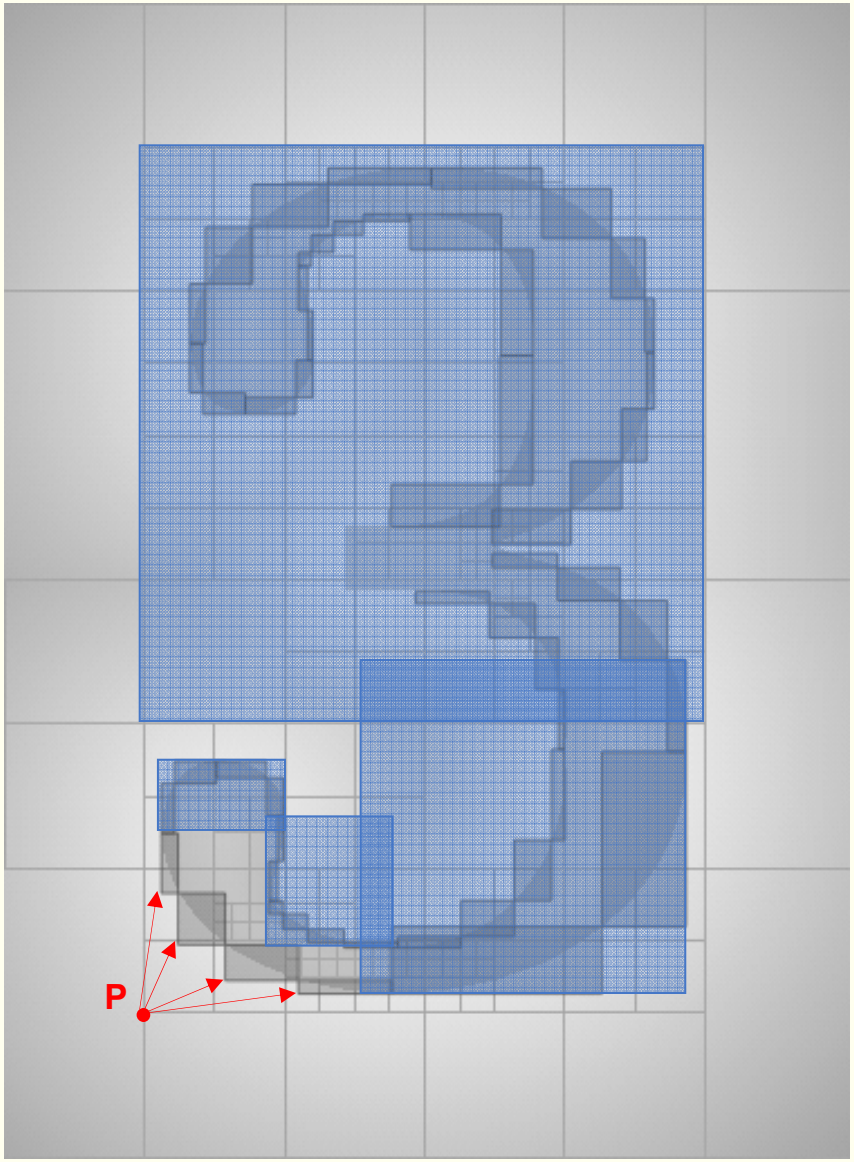
- Most of these computations are unnecessary



Cluster Trees

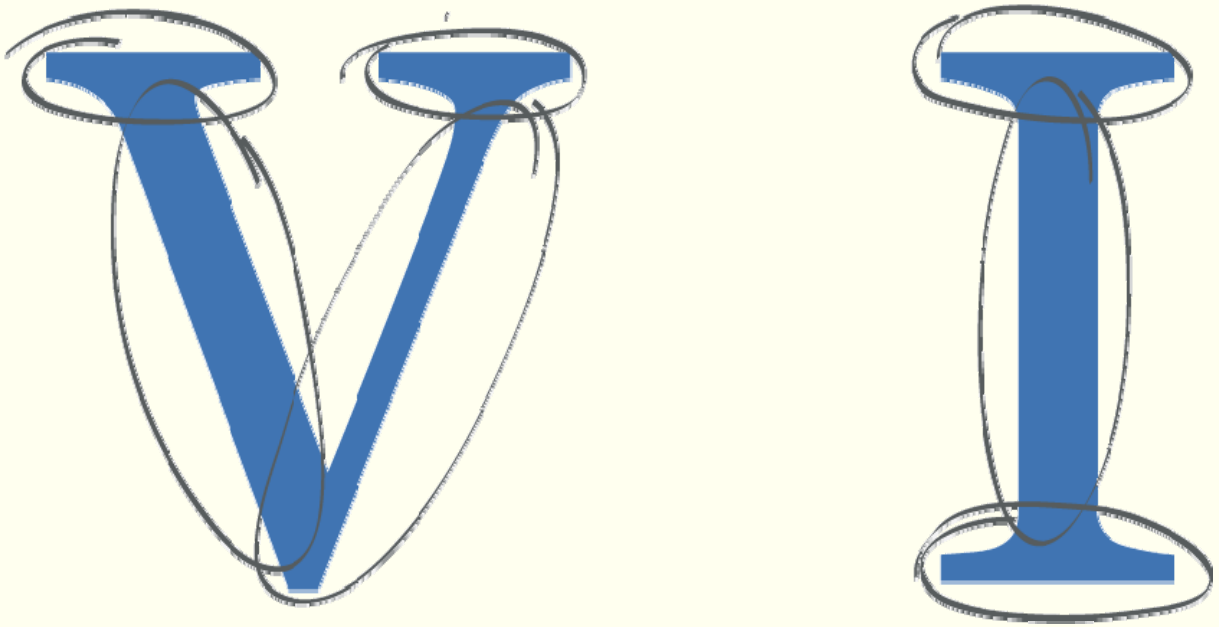
Hierarchical Bounding Regions

- Bounding boxes in the blue regions will not be the closest
- Need to prune unnecessary point-to-bounding box distance computations
- Organize bounding boxes into groups



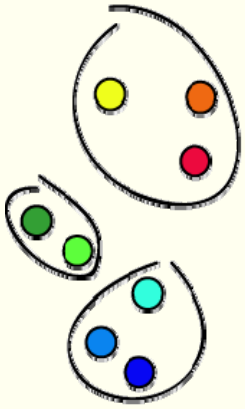
Hierarchical Bounding Regions

- Each glyph has a natural breakdown of components
- These components can be grouped hierarchically
- How do we instruct a computer to recognize the components?

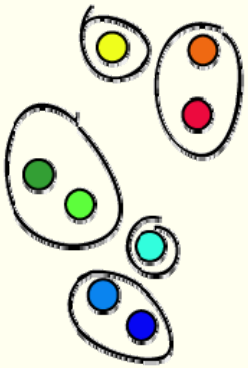


Hierarchical Bounding Regions

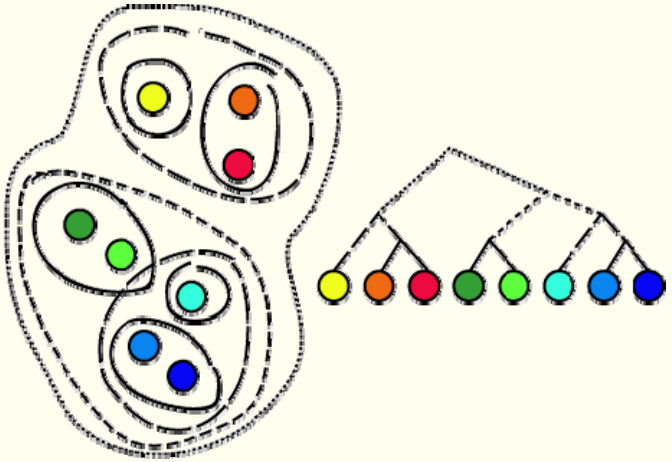
- Clustering algorithms group objects into sets
  - K-Means clustering: group objects into K separate clusters
  - Single Linkage clustering: pair objects hierarchically
- We use a hybrid
  - Cluster into groups (not necessarily a fixed number)
  - Repeat to create hierarchical groups



K-Means Clustering
(k = 3)



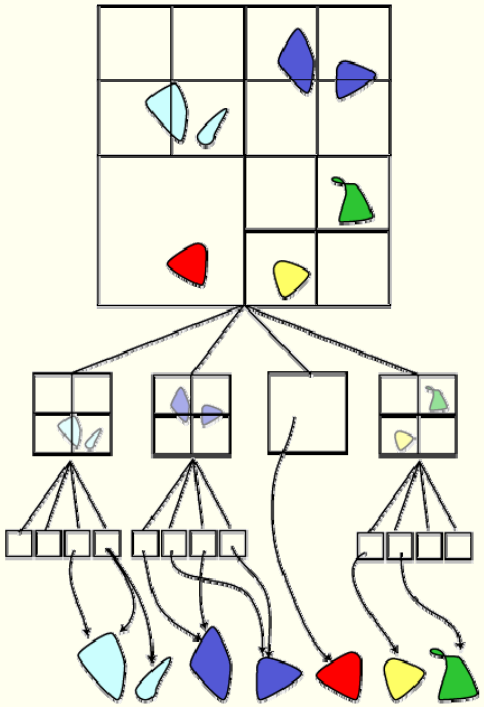
K-Means Clustering
(k = 5)



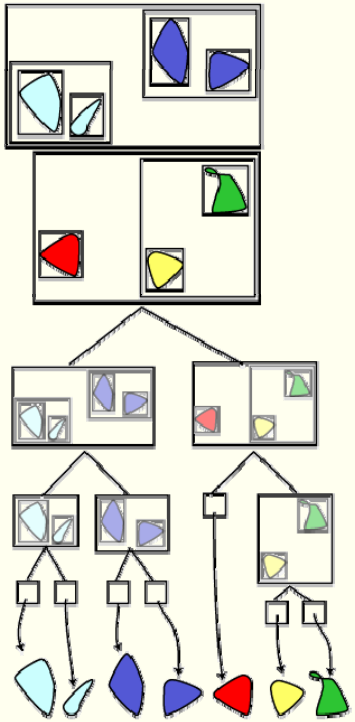
Single Linkage Clustering

Hierarchical Bounding Regions

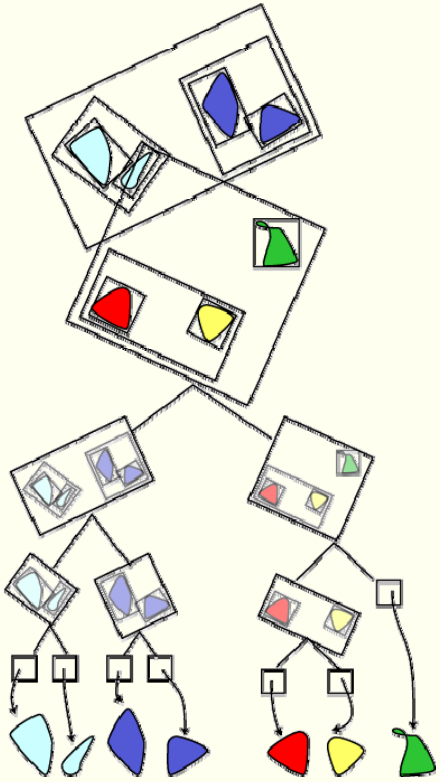
- Hierarchical spatial data structure are commonly used in computational geometry and computer graphics
- Successively break up space and store information associated with the space in a hierarchical structure



Quadtree



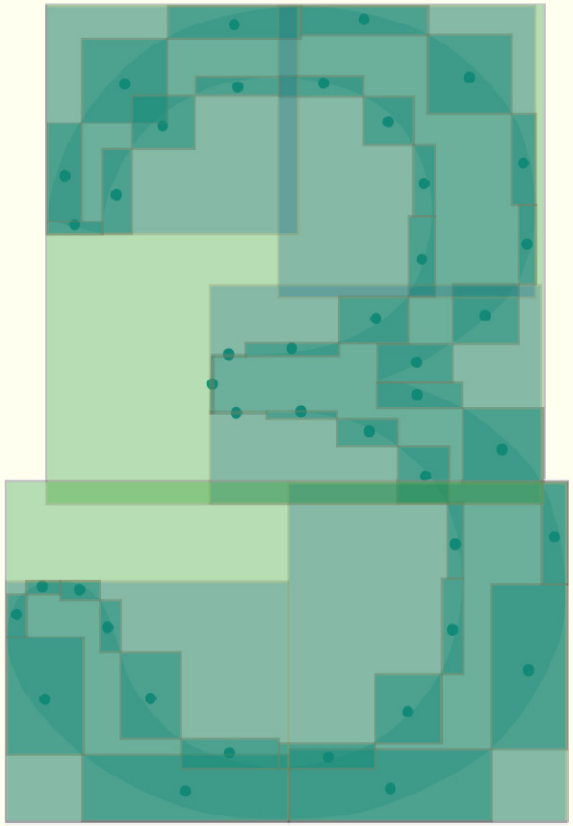
Axis-Aligned Bounding Box Tree

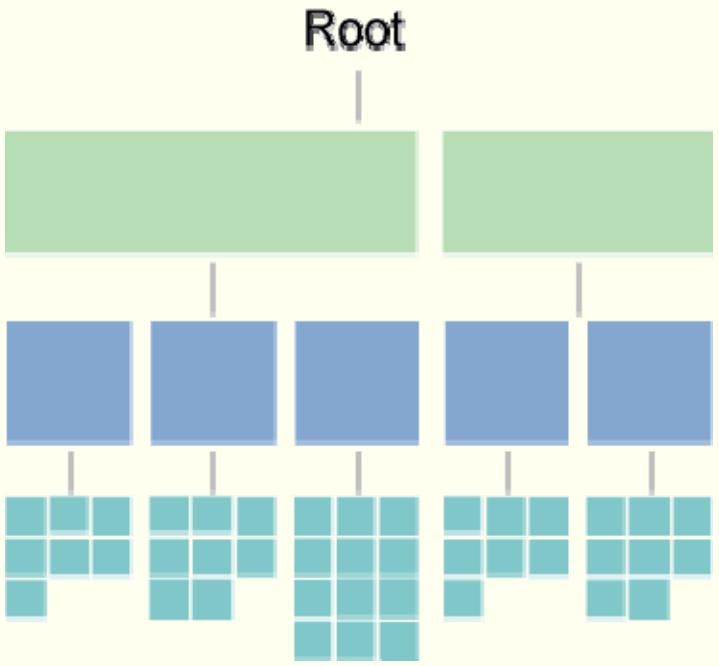


Oriented Bounding Box Tree

Building a Cluster Tree

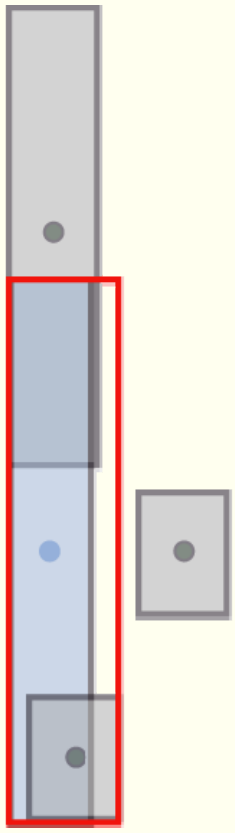
- We create a Cluster Tree of the axis-aligned bounding boxes



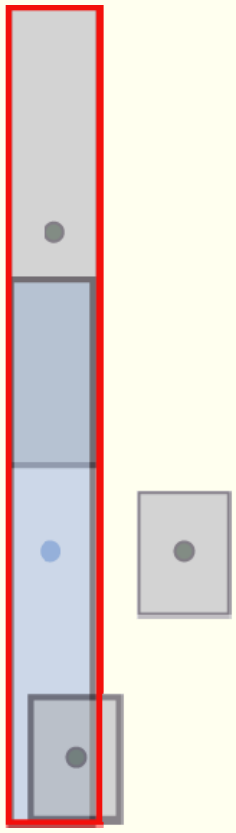


Building a Cluster Tree

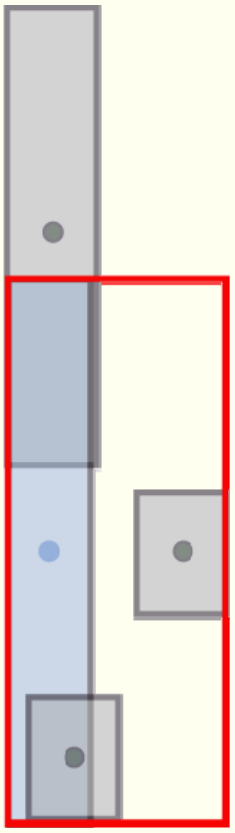
- Clustering can be based on different criteria
- We had best results clustering by minimum distance



Minimum Total Area



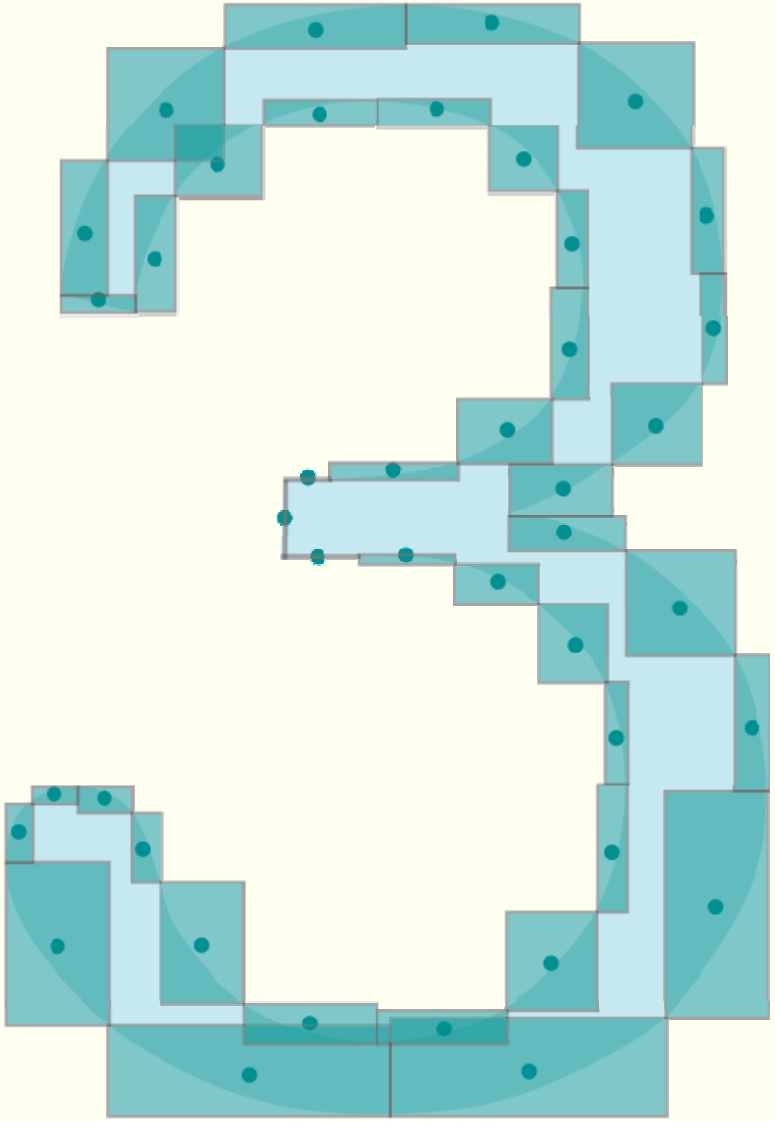
Minimum New Area



Minimum Distance

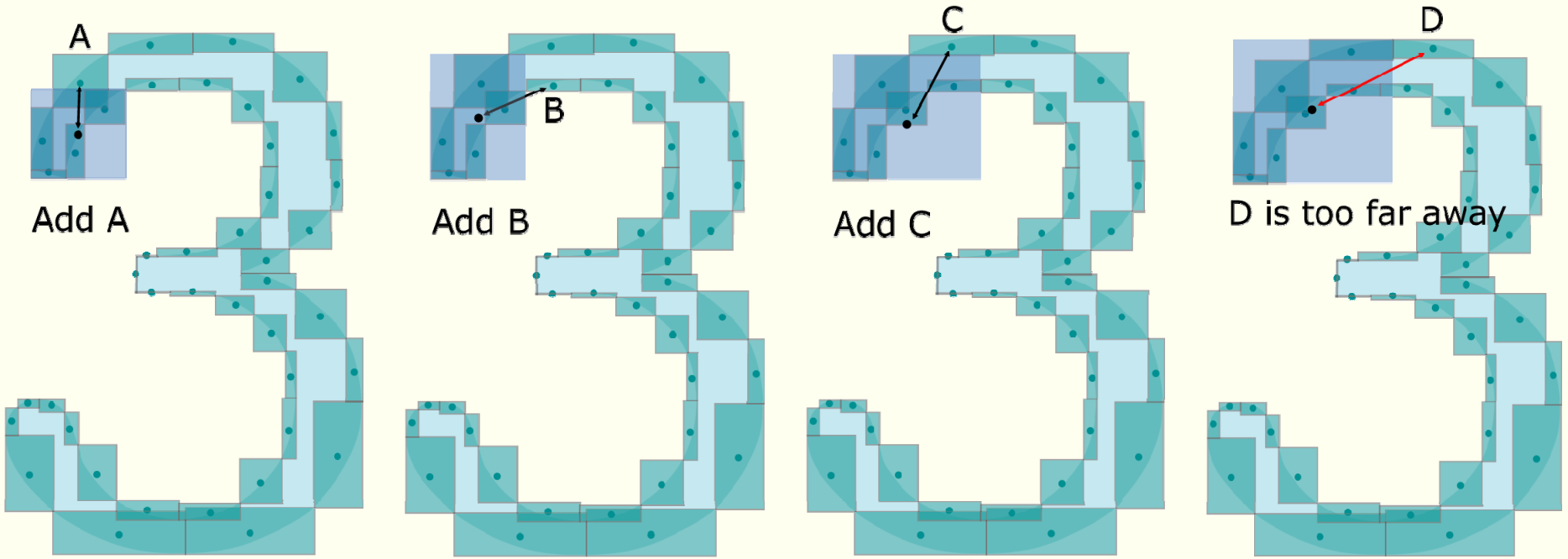
Building a Cluster Tree

- The axis-aligned bounding boxes form an initial set of leaf nodes of the Cluster Tree
- Cluster these leaf nodes into groups by proximity



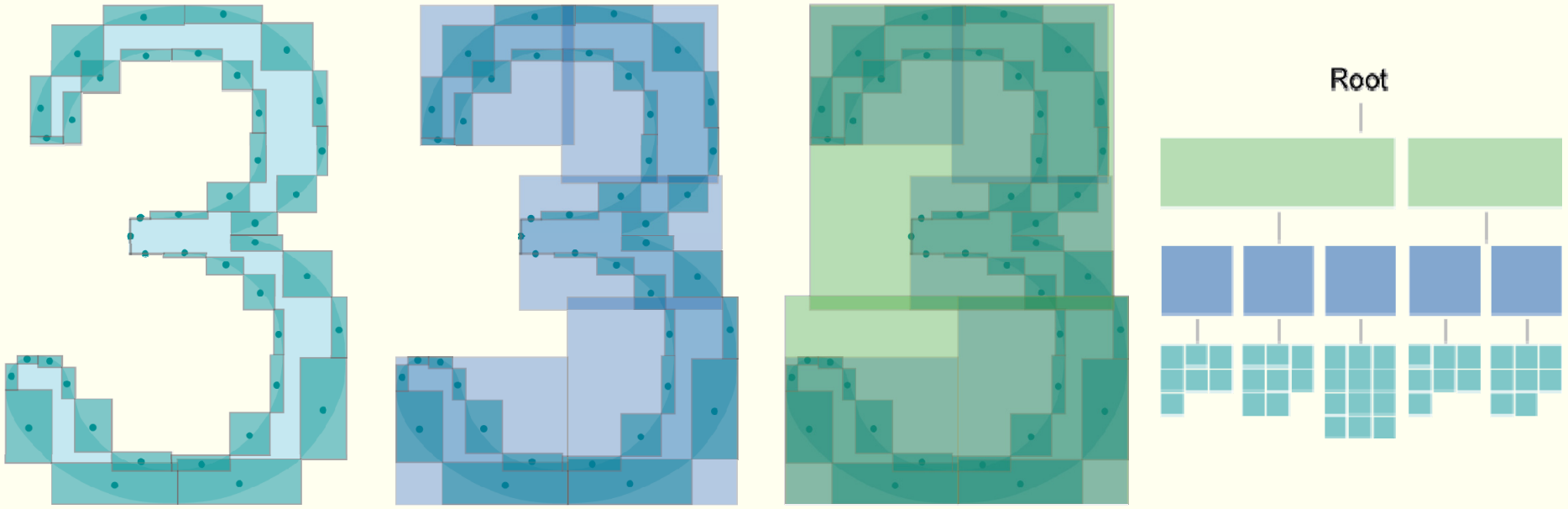
Building a Cluster Tree

- Leaf nodes whose bounding box centers are closer than a **maximum pairing distance** are clustered into groups
- The bounding boxes of these groups form intermediate nodes which are recursively clustered
- The maximum pairing distance is increased at each recursion level



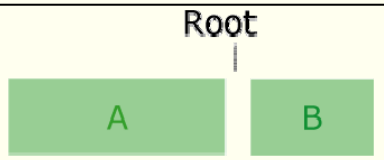
Building a Cluster Tree

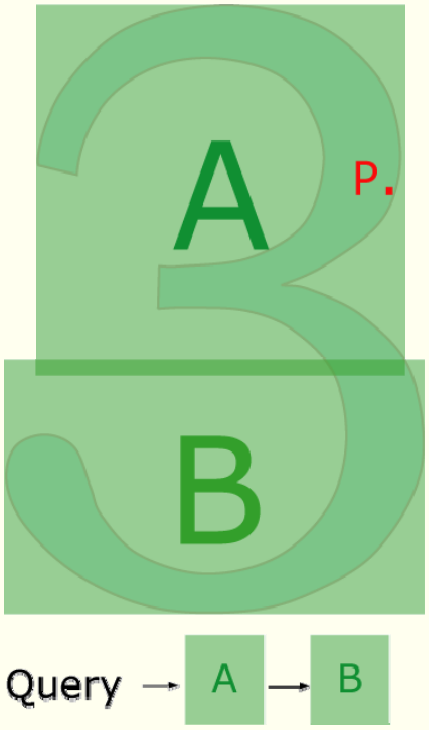
- First level of recursion: leaf nodes are clustered into five intermediate nodes
- Second level of recursion: five nodes are clustered into two new intermediate nodes, which become the children of the root node



Querying a Cluster Tree

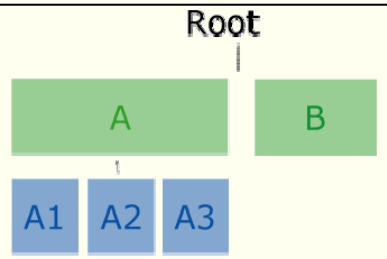
- Sort the top level nodes by minimum distance to P

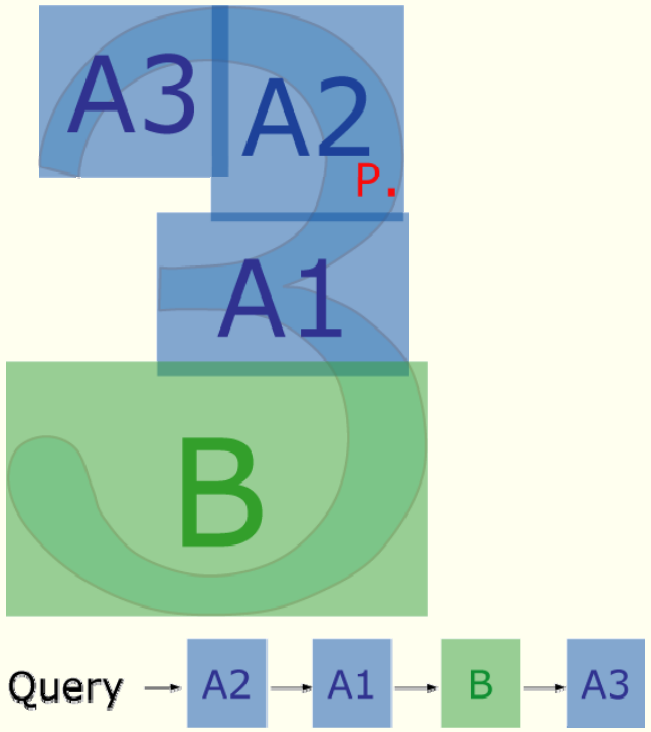




Querying a Cluster Tree

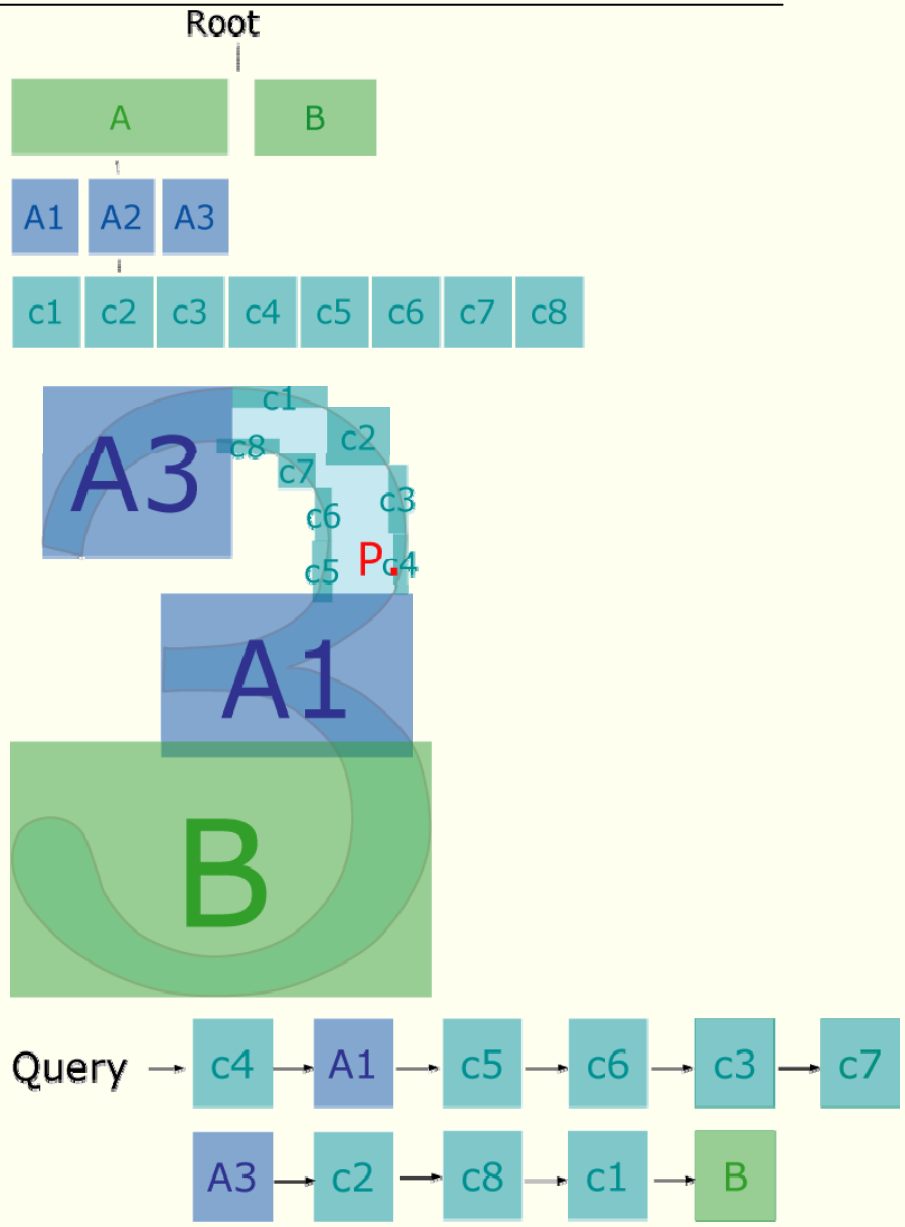
- Remove the first node from the list
- Place its contents back into the sorted list





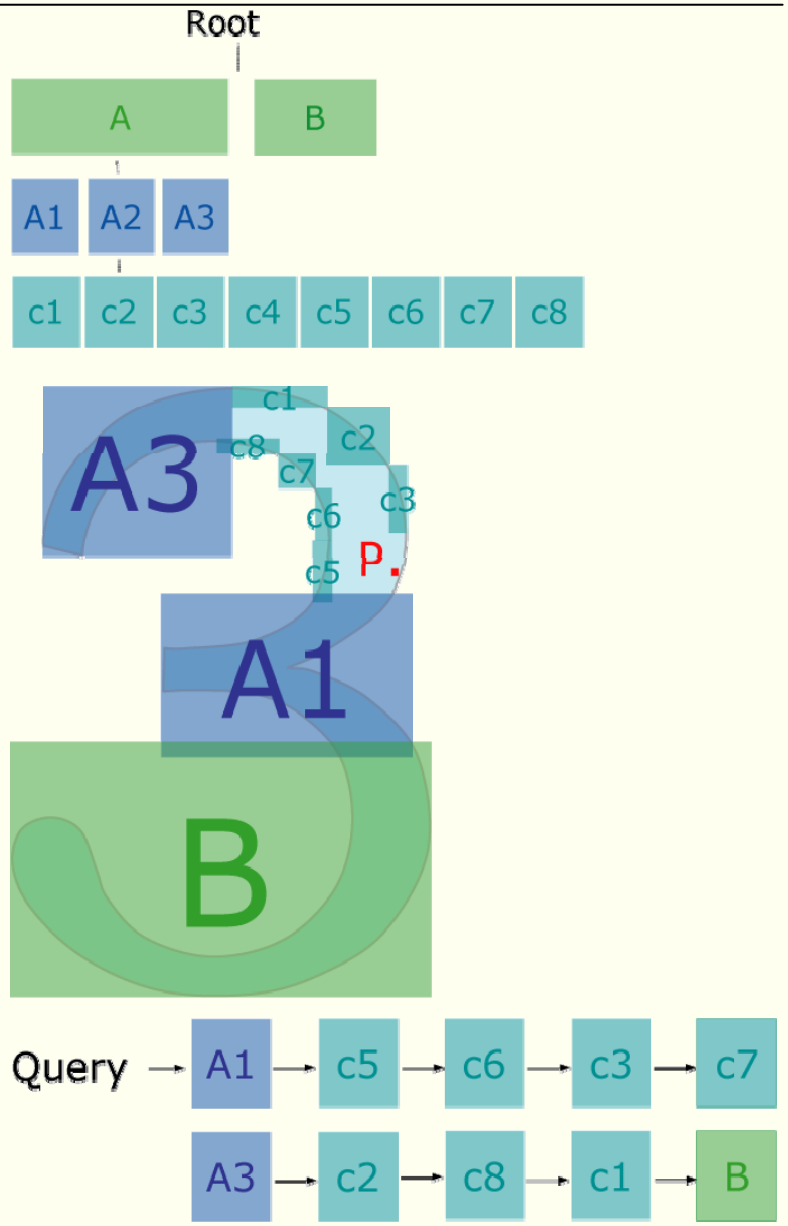
Querying a Cluster Tree

- Remove the first node from the list
- Place its contents back into the sorted list



Querying a Cluster Tree

- Remove the first node from the list
- Compute the distance from P to the Bezier curve contained in this node
- Repeat until the current distance from P to the glyph is less than the distance from P to the first node in the list



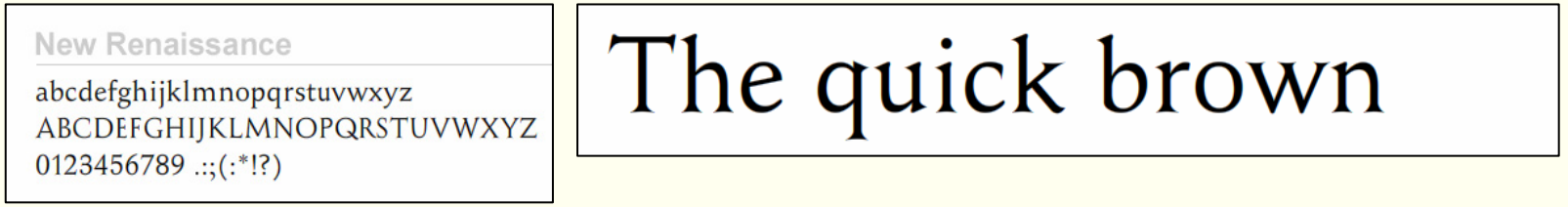
Querying a Cluster Tree

- We achieved 10x!

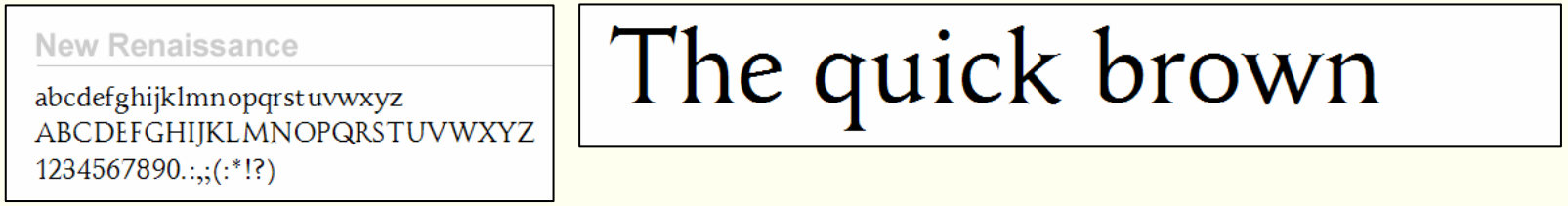
  Note: Arial is has a high proportion of straight lines so it was already fast

Font	Method	Glpyhs / Sec	Conics / Distance call
Arial	Naive	107.91	13.92
	Bounding Boxes	583.54	1.13
	Cluster Tree	575.06	0.7
Georgia	Naive	37.67	28.65
	Bounding Boxes	279.08	1.41
	Cluster Tree	413.04	0.91

Saffron

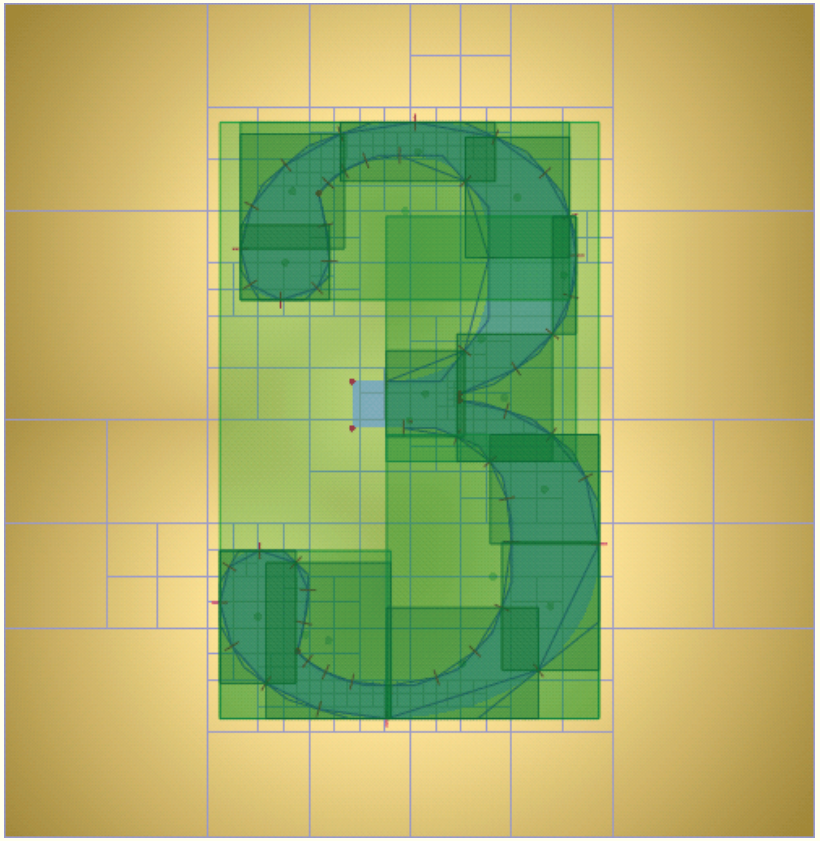


Microsoft ClearType



Conclusions & Future Work

- We can now render clear type, even at small sizes, in real-time.
- Distance based fonts, using Cluster Trees, will be in the next release of Macromedia's Flash Player!
- Outline based fonts require a fair amount of memory to store outlines, so we next investigate stroke-based fonts
- This research was done under the guidance of Dr. Sarah Frisken and Ronald Perry at the Mitsubishi Electric Research Laboratories
- Special thanks to Sarah Frisken, Ronald Perry, Diane Souvaine and George Preble for their help and guidance!



Bibliography

Bibliography

- The freetype auto-hinting resources pages.
- Heinz Breu, Joseph Gil, David Kirkpatrick, and MichaelWerman. Linear time euclidean distance algorithms. *IEEE Trans. Pattern Anal. Mach. Intell.*, 17(5):529–533, 1995.
- Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The r*-tree: an efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, pages 322–331. ACM Press, 1990.
- Olivier Cuisenaire. *Distance Transformations: Fast Algorithms and Applications to Medical*
- *Image Processing*. PhD thesis, Universite Catholique de Louvain, Louvain-la-Neuve, Belgium,
- October 1999.
- Gerald Ferin. Shape. In *Mathematics Unlimited – 2001 and Beyond*, pages 463–466. Springer-Verlag, 2001.
- Feiner Foley, van Dam and Hughes. *Computer Graphics Principles and Practice*. The Systems Programming Series. Addison-Wesley Publishing Company, 1990.
- Jr. F.S. Hill. *Computer Graphics using OpenGL*. Prentice Hall, Upper Saddle River, NJ, 2001.
- S. Frisken and R. Perry. Adaptively sampled distance fields. *Course Notes, SIGGRAPH 2001*, 2001.
- Dan Garcia. Scan conversion distillation.
- S. Gottschalk, M. C. Lin, and D. Manocha. Obbtree: a hierarchical structure for rapid interferencedetection. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 171–180. ACM Press, 1996.
- Antonin Guttman. R-trees: a dynamic index structure for spatial searching. In *Readings in database systems*, pages 599–609. Morgan Kaufmann Publishers Inc., 1988.
- Roger D. Hersch, Claude Btrisey, Justin Bur, and Andr Grtler. Perceptually tuned generation of grayscale fonts. *IEEE Comput. Graph. Appl.*, 15(6):78–89, 1995.

Bibliography

- Roger D. Hersch. Font rasterization: the state of the art. In *Visual and technical aspects of type*,
- pages 78–109. Cambridge University Press, 1993.
- Changyuan Hu and Roger D. Hersch. Parameterizable fonts based on shape components. *IEEE Comput. Graph. Appl.*, 21(3):70–85, 2001.
- Erik G. Hoel and Hanan Samet. A qualitative comparison study of data structures for large line segment databases. In *Proceedings of the 1992 ACM SIGMOD international conference on Management of data*, pages 205–214. ACM Press, 1992.
- A. Henrich, H. W. Six, and P. Widmayer. The lsd tree: spatial access to multidimensional and non-point objects. In *Proceedings of the fifteenth international conference on Very large data bases*, pages 45–53. Morgan Kaufmann Publishers Inc., 1989.
- David E. Johnson and Elaine Cohen. A framework for efficient minimum distance computations. 1998.
- M. W. Jones and R. A. Satherley. Shape representation using space filled sub-voxel distance fields. In *Proceedings of the International Conference on Shape Modeling & Applications*, page 316. IEEE Computer Society, 2001.
- F. Leymarie and M. D. Levine. Fast raster scan distance propagation on the discrete rectangular lattice. *CVGIP: Image Underst.*, 55(1):84–94, 1992.
- Saffron type system, 2004.
- Perry and Frisken. Kizamu: A system for sculpting digital characters. *Proc. SIGGRAPH 2001*, pages 47–56, 2001.
- Dan Pelleg and Andrew Moore. *X*-means: Extending *K*-means with efficient estimation of the number of clusters. In *Proc. 17th International Conf. on Machine Learning*, pages 727–734. Morgan Kaufmann, San Francisco, CA, 2000.
- Bradley A. Payne and Arthur W. Toga. Distance field manipulation of surface models. *IEEE Comput. Graph. Appl.*, 12(1):65–71, 1992.

Bibliography

- S. Quinlan. Efficient distance computation between non-convex objects. In *IEEE Intern. Conf. on Robotics and Automation*, pages 3324–3329. IEEE, 1994.
- A. Ricci. A constructive geomtery for computer graphics. *Computer Journal*, 16(2):157–160, 1973.
- Azriel Rosenfeld and John L. Pfaltz. Sequential operations in digital picture processing. *J. ACM*, 13(4):471–494, 1966.
- Hanan Samet. The quadtree and related hierarchical data structures. *ACM Comput. Surv.*, 16(2):187–260, 1984.
- Hanan Samet. Hierarchical representations of collections of small rectangles. *ACM Comput. Surv.*, 20(4):271–309, 1988.
- Sergei Savchenko. *3D Graphics Programming, Games and Beyond*. Sams Publishing, Indianapolis, Indiana, 2000.
- J. A. Sethian. A fast marching level set method for monotonically advancing fronts. In *Proceedingsof the National Academy of Sciences*, volume 93, pages 1591–1595, 1996.
- R. Perry S. Frisken and T. Jones. Adaptively sampled distance fields: A general representation of shape for computer graphics. *Proc. SIGGRAPH 2000*, pages 249–254, 2000.
- Angel Domingo Sappa and Miguel Angel Garcia. Hierarchical clustering of 3d objects and its application to minimum distance computation. In *IEEE Intern. Conf. on Robotics and Automation*, pages 5287 – 5292. IEEE, 2004.
- R. Satherley and M. W. Jones. Vector-city vector distance transform. *Computer Vision and Image Understanding*, 82:238–254, 2001.
- Hanan Samet and Robert E. Webber. Storing a collection of polygons using quadtrees. *ACMTrans. Graph.*, 4(3):182–222, 1985.
- Bob Thomas. Font fusion: Technology for making text look good anywhere.
- Gino van den Bergen. Efficient collision detection of complex deformable models using aabb trees. *J. Graph. Tools*, 2(4):1–13, 1997.

Bibliography

- John E. Warnock. The display of characters using gray level sample arrays. In *Proceedings of the7th annual conference on Computer graphics and interactive techniques*, pages 302–307. ACM Press, 1980.
- Douglas E. Zongker, Geraldine Wade, and David H. Salesin. Example-based hinting of true type fonts. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 411–416. ACM Press/Addison-Wesley Publishing Co., 2000.