Computer-Aided Design and Applications

http://www.cadanda.com

Point-Distance Computations: A Knowledge-Guided Approach

Les A. Piegl, Khairan Rajab, Volha Smarodzinava and Kimon P. Valavanis

University of South Florida, {lpiegl,khairanr,olyagrove,kvalavanis}@gmail.com

ABSTRACT

A knowledge-guided approach is presented to compute the distance between a point and NURBS curves and surfaces. Given a set of query points, a global algorithm is introduced that finds the required distances. The method has the following major components: (1) a discrete version of a physical wave model is implemented to find nearest regions, (2) a fast parametrization-based decomposition is used to localize the search for start points, (3) a spatial data structure is employed for fast nearest neighbor searching, and (4) knowledge is injected into the process to increase the robustness of the error-prone numerical process.

Keywords: minimum distance computation, robustness, knowledge-guided systems, NURBS.
DOI: 10.3722/cadaps.2008.855-866

1. INTRODUCTION

Computing the minimum distance between a point and objects used in various fields of computer assisted design has been a long standing, and by and large, unsolved problem. The majority of the issues stem from the uncertainty of the numerical process that relies on good start points (that are not easy to find) and expensive iterations (that may not converge). The emergence of point-based modeling, e.g. in bio-engineering or garment design, renewed interest in researching more reliable methods for distance computation. Traditionally, distances were used for collision detection, path planning, interference checking, assembly management or quality assurance, just to name a few. Applications in biology, medicine or fashion brought new challenges as the point cloud, obtained from various forms of scanners, may contain a fairly large amount of data. As the only point of reference/origin to the (human) model is the point cloud itself (it was not modeled by a CAD system), checking, say, the best fit of a pair of sunglasses, requires lots of point distance calculations between the CAD design and the human head point cloud model. The challenges come from the large amount of data to be processed and the sophistication of the design, i.e. a fancy sunglasses model with intricate details.

This paper presents a method to solve the following problem. Given a set of points $P_i, i = 0,...k$, and a NURBS curve $C(u)$ or surface $S(u,v)$, compute the distances $d_i[P_i,C(u)]$ and $d_i[P_i,S(u,v)]$ to within a required accuracy. The method presented herein contains the following major components:

- A knowledge-guided formulation [17] that increases robustness, reliability and speed. Information, stored in the knowledge base of a knowledge-guided NURBS kernel, is used to make intelligent decisions to get start points, to avoid convergence failures and to speed up the computation via special casing.
- A wave model-based definition of point distance and its discretized implementation.
- A fast and parametrization-based decomposition of NURBS entities to aid in the localization of appropriate start points.
- A spatial data structure for fast nearest neighbor searching and to accommodate multiple distance computations without having to discretize the geometry more than once.

The solution is global, i.e. no start/guess points are provided. It is the belief of the authors that the critical element of distance computation is data preparation and start point selection for the iterative process. How fast and how reliably the numerical process works depends nearly 100% on how well the data is prepared. To this end the following operations need to be performed:

- Data validation, e.g. the number of knots and control points must satisfy the required conditions.
- Data cleaning, e.g. unnecessary knots and control points must be removed, the curve must be decomposed so that the segments do not contain cusp or loops, etc.
- Data purging, e.g. large amount of control points/knots must be reduced as knots in close proximity can cause numerical failures.

Experience has taught us that feeding a NURBS curve or surface blindly into numerical algorithms causes failures almost certainly. We cast our lot to knowledge-guided systems that allow the application programmer to make intelligent decisions and to make more robust use of numerical algorithms.

Prior work on point distance computation has not been as extensive as the complexity of the problem would justify. The majority of the references are in the field of robotics [4-6,14,15,23,24] (collision detection) and computer vision [1,20] (registration). Special surfaces, e.g. canal surfaces, are singled out as they provide algebraic, closed form solutions [2,10-13,22]. A few methods are available for NURBS [3,8,16,18,21], most of which employ control polygon/net based subdivision and segment elimination [9,11]. While control polygon/net based methods come natural in a NURBS-based environment, they work with flatness conditions that may not assist numerical iterations that are inherently parametrization dependent.

The organization of the paper is as follows. Section 2 overviews some components of KGnurbs™, a knowledge-guided NURBS kernel. In section 3 a summary of point distance formulas are given along with an analysis of major pitfalls. Section 4 presents the wave model used to find the shortest distance and in section 5 curve and surface decompositions are introduced without using the control polygons/nets. The main algorithm is outlined in section 6 followed by tests and examples in section 7. Some conclusions are provided at the end of the paper.

2. KNOWLEDGE-GUIDED NURBS

Knowledge-guided NURBS (KGN) [17] were introduced recently to address the raising concern about robustness and reliability of CAD systems, data transfer, compatibility, integration and better design quality. The main components of KGN are depicted in Fig. 1 below.

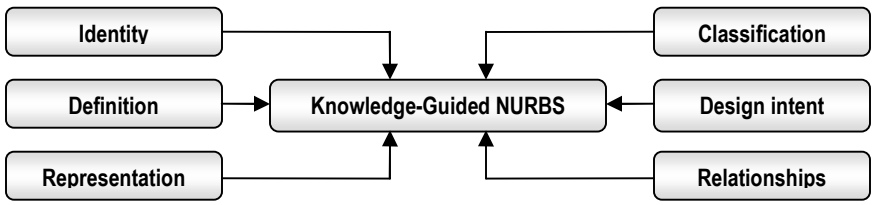


Fig. 1: Components of knowledge-guided NURBS.

Each NURBS entity consists of the following elements:

- **Identity** provides proper *naming* and *identification*.
- **Definition** means *control points*, *knots* and *degree*(s).
- **Representation** delivers information about the quality of the *parametrization*, the levels and types of *continuity* and any or all forms of *irregularities*, e.g. cusp, zero curvature, etc.
- **Classification** tells the application programmer about the *type* of the entity, its *origin* and its intended *destination* or possible applications.
- **Design intent** is most useful to know what *functions* were used to design the entity, what *alternatives* were considered and what *decisions* were made.

- **Relationship maps** are complicated data structures that store object *references*, *relationships* and their *parameters*. For example, when a NURBS circle is computed, the NURBS and the conic enter into a *definition* relationship and all the parameters needed to recreate the design are saved.

Knowledge-guided NURBS are extensions of traditional NURBS in that they support design changes, data transfer, design replay, and what is most relevant in this paper, more robust computations. To enhance the robustness of distance computations, KGN is utilized in a number of different ways:

- Certain *object types* are handled by special algorithms. Lines and circles are handled separately for curves and planes, spheres, cones, cylinders and torii for surfaces. Other types of surfaces that may be considered separately are ruled surfaces and surfaces of revolution.
- *Relationship maps* are utilized in two ways: (1) to obtain the parameters of the original construction, e.g. once it is determined that the curve type is circle, the center, the radius and the sweep angle are obtained from the knowledge base, and (2) to glean an insight into the design process, e.g. the circle may be obtained via definition or via fitting to a random set of points.
- Information on object *representation* is most useful in numerical processes like distance computations. The following measures are utilized:
    - Parametrization factor: an indicator that tells the system how far the curve is from being uniformly parametrized. This becomes critical in finding the proper decomposition, i.e. the iterative process does not "run-off" in case of highly non-uniform parametrization.
    - Level of continuity: it is useful to know if the curve is G- or C-continuous and at what level. Many numerical algorithms require first or second degree parametric continuity.
    - Irregularities: typical problems that hinder numerical code from working are vanishing derivatives and discontinuities, e.g. cusps, poles, seams or collapsing derivatives for surfaces. If the knowledge is made available to the application programmer, robustness can be increased by special code segments that account for these irregularities.

The algorithm presented below takes advantage of all of the above. The good news is that the robustness has been increased considerably; the bad news is that it comes at a price of a very large code size. It requires a significant chunk of an entire NURBS kernel supported by a sophisticated knowledge base.

3. DISTANCE FORMULAS

The basic formulas for distance computations are quite simple [18]. For curves we have:

$$f(u) = (P - C(u)) \cdot C'(u) = 0$$

where P is the given point and a point on the curve $C(u)$ is sought at some parameter u_0 so that the above equation is satisfied. A common solution is to use Newton's iteration, with a proper start parameter, to solve the univariate functional equation $f(u) = 0$. The iteration is halted if (1) the point is near the curve, or (2) $P - C(u_0)$ and $C'(u_0)$ are nearly perpendicular, or (3) consecutive parameters do not change much. The method fails to converge under the following circumstances:

- The start guess parameter is too far from the root and (1) the allowed iterations are not enough to reach convergence, or (2) the new parameters jump too much and the iterative process simply gets lost. Remedy: find a better start value.
- The curve is closed and the newly computed parameter is out of the domain. Remedy: bring the parameter back at the other end of the domain. That is, if the domain is $[u_L, u_R]$ and the new parameter $u_{i+1} > u_R$, then reset the parameter as $u_{i+1} = u_L + (u_{i+1} - u_R)$.
- The curve is open and the query point "falls off" at the end. In this case Newton is going to push the parameters out of the domain. Remedy: clamp the parameters and stop iterations. The closest point is one of the end points.
- The curve has a discontinuity, e.g. a cusp, and the closest point is at the point of discontinuity. In this case the Newton method tends to jump over the discontinuity or hovers around it, entering into an infinite loop, if it is the end point of a closed curve. Remedy: preprocess the curve into continuous pieces and process them individually.

For surfaces there are two equations to be solved:

$$f(u,v) = (P - S(u,v)) \cdot S_u(u,v) = 0$$
$$g(u,v) = (P - S(u,v)) \cdot S_v(u,v) = 0$$

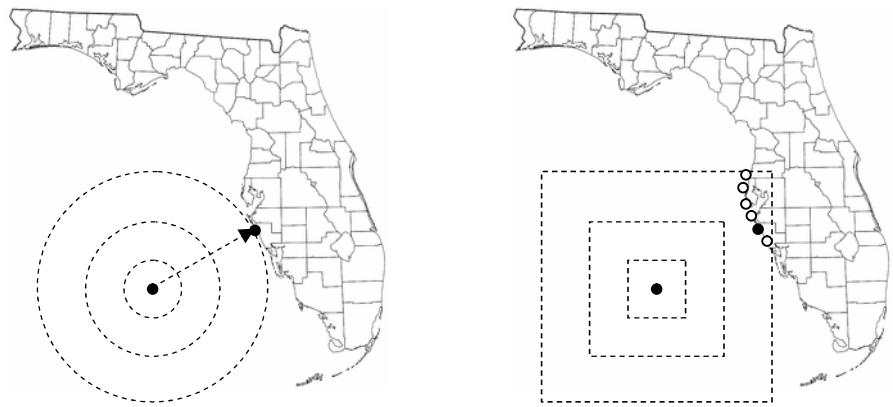
The solution is at the point (u_0, v_0) where the vectors $P - S(u_0, v_0)$ and $S_u(u_0, v_0)$, and $P - S(u_0, v_0)$ and $S_v(u_0, v_0)$ are perpendicular. Using multivariate Newton iteration, the process is stopped when the point is found to lie on the surface, the above vectors are perpendicular, or the parameters do not change. The common pitfalls are the same as those for curves, plus two more:

- The numerical process for surfaces is much more dependent on the choice of the start point. The presence of the u- and v-directional parametrizations (with independent knot vectors) makes it difficult for the algorithm to converge if one (or both) of the parametrizations is far from being uniform. Remedy: choose the start point with a higher level of accuracy (decomposition) than for curves.
- If the nearest point happens to be one of the boundary points, i.e. the perpendicular projection of the query point is off the patch, the surface algorithm will enter into an infinite loop. This is because the simultaneous perpendicular conditions above cannot be satisfied in at least one direction. Remedy: extract the boundary and use the curve algorithm to find the nearest point on the boundary.

While the mathematics of distance computation is very simple, implementation of the formula into a robust code is a highly non-trivial task. Newton method works at its best when the start parameter is near the solution and the NURBS entities are smooth and free from irregularities. Data preparation is the key ingredient here and the more knowledge is available the better job the preprocessor does on slicing up the NURBS entities for failure free distance computation.

4. THE WAVE MODEL

Assume that a fisherman is out fishing in the Golf of Mexico, off the coast of South Florida. He receives a warning on his marine radio that a hurricane is heading to the Golf and it is advisable to come ashore as fast as possible. Theoretically speaking, he may find the shortest distance to the West Coast as follows: he drops a big rock in the water and follows the path of the circular waves the rock creates, Fig. 2(a). The point where the wave hits the first time is the closest point defining the straight course that gets him to safety the fastest possible way. That is, to find the shortest distance between a point and a NURBS object (the shoreline curve in our example), all needs to be done is to propagate a circular wave and find the point of first hit. Although this is a simple idea, the implementation may not be that efficient.



(a) (b)

Fig. 2: The wave model to find the closest point: (a) the physical model, (b) its discretized implementation.

Let us reconsider the idea from the point of view of spatial orientation (subdivision) and efficient implementation. Assume that the wave can take on a shape of a rectangle, Fig. 2(b). As the rectangle grows bigger, it will hit the shore either at a vertex or along an edge. Because this may not provide the shortest distance, we allow the wave to penetrate into the shoreline a certain distance to get a piece that contains the location of the endpoint of the shortest vector (empty circles mark the candidate area in Fig. 2(b)). A discrete sampling along the candidate area provides a good start point for iteratively, via Newton, finding the foot-point of the shortest distance.

To make this idea a useful algorithm, the following details need to be worked out:

- Decomposition of the NURBS entities for fast localization and to accommodate multiple point distance computations.
- Spatial subdivision and object binning to hold the data for fast range searching (nearest neighbor finding).
- Finding start points for fast Newton-type iteration using the wave model just described.

The sections below elaborate on the relevant details.

5. OBJECT DECOMPOSITION

In the world of NURBS, decomposition almost always triggers the process of subdivision [9,16,20], i.e. splitting the objects into smaller segments until they are close enough to a line (curve case) or to a quadrilateral (surface case). While NURBS subdivision is a nice geometrical technique, it has several drawbacks when used for closest point computation:

- It is not cheap as computations have to done on the control points in 3-D. The method proposed below works primarily with 1-D B-splines.
- Subdivision algorithms require sophisticated memory management (a dynamic stack needs to be managed holding intermediate results) which is time consuming.
- For good start points a fairly fine subdivision is required, which is fine if segments that are known not to contain the closest point can be eliminated. However, if there are a large number of query points, possibly randomly distributed, the subdivision may have to produce a fine polygonal approximation.

The method presented below is inexpensive and requires no complicated memory management. It is also easily adjustable based on the information available in the knowledge base. The main idea comes from many years of experience with Newton-type methods that worked quite well for curves and surfaces that were uniformly or nearly uniformly parametrized.

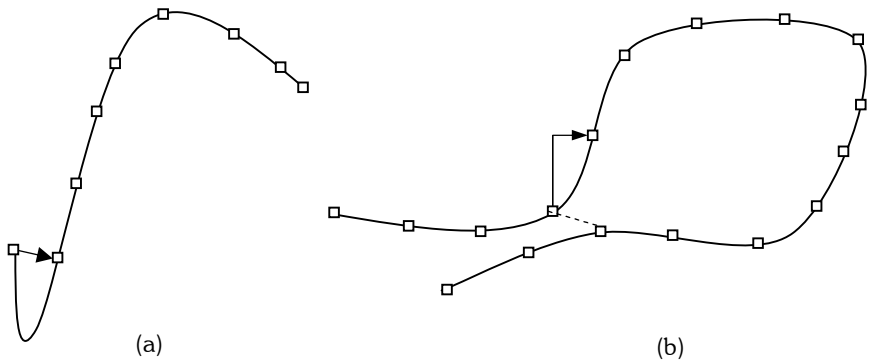


Fig. 3: Decomposition problems: (a) small hook at the start and tiny segment at the end; (b) curve comes close to itself does not pose a problem.

NURBS curves are decomposed into segments by computing a nearly uniform point distribution as follows. Given a tolerance tol which is a percentage of the arc length, and a jitter factor jit , the algorithm computes a nearly uniform point distribution to satisfy the following inequality:

$$\varepsilon_l < \left|C(u_i) - C(u_{i-1})\right| < \varepsilon_u \quad \varepsilon_l = jit \cdot tol \quad \varepsilon_u = tol$$

If $jit \approx 1.0$ the distribution is almost uniform, and if $jit \approx 0.0$ it is random within the allowed tolerance. In most of our tests we used $tol = 5\%$ and $jit = 80\%$. The major components of the algorithm are as follows:

1. Get the upper bound as the required percentage of the approximate arc length, and the lower bound by jittering the upper bound.
2. Get the first guess parameter by dividing the parametric domain with the expected number of points, e.g. if the upper tolerance is 5%, the expected number of points is 20.
3. Apply a binary search type method to find the first parameter for which the above inequality holds. Get the first parameter increment as the difference between the found parameter and the left knot.

4. Increment the found parameter with the difference and bring the new guess point into compliance with the inequality applying the same binary search type method.
5. The new guess parameter at any given point is obtained by incrementing the current parameter with the difference between the current and the previous.
6. The algorithm stops when the end point of the curve is reached.

This is a very simple method that requires point evaluations only which needs about a third of the computations of subdivision. It is also fairly efficient as it relies on *spatial coherence*, i.e. there is not much change from segment to segment. That is, the guess parameter needs to be adjusted only occasionally. Practical experience shows that computing N nearly uniformly spaced points requires about $1.25 \cdot N$ point evaluations for most practical curves.

Fig. 3(a) shows two slight drawbacks of the method: a small hook at the beginning and a tiny segment at the end. The small hook produces a segment that is somewhat longer than expected and the tiny segment is somewhat shorter. Neither one of these impact the algorithm adversely because the upper tolerance is conservatively chosen and the segments will be further decomposed to find a good start parameter for Newton iteration. A seemingly problematic case is depicted in Fig. 3(b), where the curve comes close to itself. Because the curve is topologically equivalent to a line and because the stepping is done in the parameter space, no jump to the other part of the curve is made (dashed line). The computed points will sweep out the curve irrespective of how close it comes to itself.

NURBS surfaces use curve decomposition to generate a grid of points. First, the surface is refined so that the control net is close to the surface (the approximate longest extents in both directions do not change more than a given percentage). Then the longest iso-curves (computed at the nodes of the refined control net [18]) are extracted and the curve algorithm above is applied to get a nearly uniform point distribution. The parameters of the corresponding points are then used to get a grid of points on the surface. Two difficult examples are shown in Fig. 4 where the crosses mark decomposition points. Both the dust pan (left) and the paint brush handle (right) have highly varying curvatures and dimensions. Since the sampling relies on the longest iso-curves, some areas will be sampled as desired, others will be over-sampled. Fortunately, this is not a problem either for the sampling routine (the computation is proportional to sampling two curves) or for the Newton iteration that gets a better start point in the over-sampled areas.

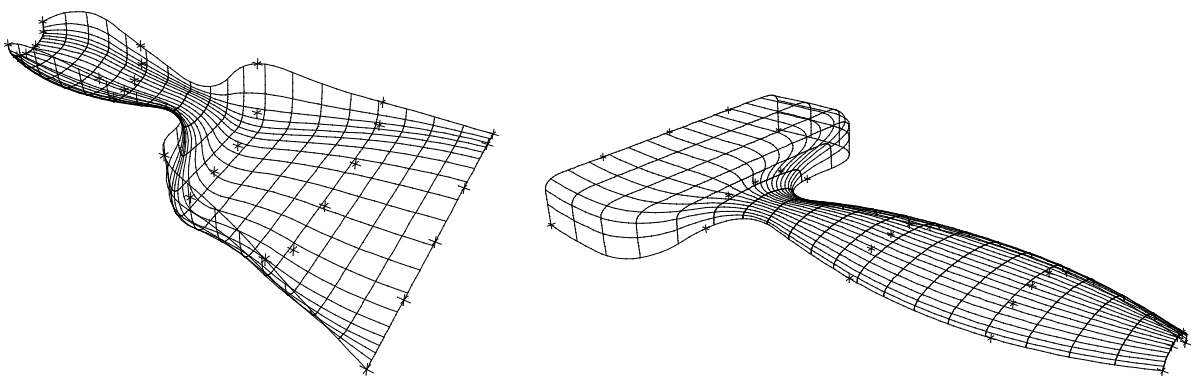


Fig. 4: Surface decomposition examples using a grid of points.

In order to make the decomposition algorithm work for practical applications, the upper bound tol and the jitter factor jit must be properly chosen. It may seem that they can be adjusted experimentally, a closer look at parametrization reveals a surprising fact. Good start points are easy to find if the curve/surface is parametrized uniformly, i.e. if the particle moves along the path of the curve/surface with constant speed while the parameter sweeps out the parametric domain. Since speed is measured by the magnitude of the first derivatives, a measure, called *parametrization factor* is introduced (and stored in the knowledge base) as follows:

$$pf = \frac{D_{\max} - D_{\min}}{D_{\max}} \quad D_{\min}, D_{\max} \text{ are min-max derivatives}$$

If the curve is parametrized uniformly $D_{\max} = D_{\min}$ and $pf = 0$. On the other hand, if the entity has a vanishing derivative (the particle comes to a stop), the parametrization factor is 1. In other words, the smaller the factor, the more uniform the parametrization. The maximum and the minimum derivative magnitudes are computed (approximately) as follows:

- Compute the derivative curve/surface using symbolic operators [19].
- The maximum and the minimum derivatives are computed from the derivative curve/surface via repeated knot refinement (one or two levels of knot refinement provides a very good estimate).

Once the parametrizaton factor has been computed, various categories can be set up based on the type of application at hand. For example, styling application designers may establish the following relationship:

Pf	Parametrization	Tolerance	Jitter
$0.0 \le pf \le 0.2$	Nearly uniform	0.2	0.5
$0.2 < pf \le 0.4$	Nearly non-uniform	0.1	0.6
$0.4 < pf \le 0.7$	Non-uniform	0.05	0.7
$0.7 < pf \le 1.0$	Highly non-uniform	0.01	0.8

Practical experience shows that Newton can always be made to work if the algorithm receives a close enough start parameter. The categories above, perhaps in a more refined format, can greatly enhance the reliability (convergence to the correct root) and the performance (optimal level of refinement) of distance routines. To illustrate that such knowledge base is necessary, the Tab. 1 below shows parametrization factors for circular arcs of various degrees.

Degrees	Pf/min-max weights	90˚ arc	120˚ arc	360˚ arc
$p = 2$	pf	0.14	0.06	0.14
	$w_{\max}$	1.0	1.0	1.0
	$w_{\min}$	0.707	0.86	0.707
$p = 4$	pf	0.005	0.007	0.007
	$w_{\max}$	1.0	1.0	1.0
	$w_{\min}$	0.93	0.89	0.89
$p = 5$	pf	0.04	0.07	0.51
	$w_{\max}$	1.35	1.43	5.0
	$w_{\min}$	1.23	1.21	1.0

Tab. 1: Parametrization factors for circles of different degrees and sweep angles.

The NURBS circle is a very disappointing entity as its parametrization is not only non-uniform, as one would rightfully expect, the parametrization factor is different from arc to arc. Here is a serious problem. Take a query point and a circular arc of 90 degrees sweep, and compute the distance between the two. Now, extend the same arc by 30 degrees and repeat the distance computation. The results will not be the same! So the conclusion is simply this: numerical processes are subject to the parametrization that has to be made available to the application process, i.e. the knowledge base must contain information on the validity of the results as a function of the parametrization factor.

6. THE MAIN ALGORITHM

Given a NURBS curve or a surface, a set of random points, a point coincidence tolerance and a perpendicularity tolerance, the algorithm outlined below computes the distances of the given points from the NURBS entities to within the required accuracy. The major steps of the algorithm are as follows:

1. Check the object classification to see if the NURBS entity is a line, circle, plane, sphere, cone, cylinder or a torus. If yes, mine the knowledge base to find how the entities were created and get the defining parameters. Use special purpose code to compute the distances between the points and the conic/quadric.
2. Check the object representation to find out if information on parametrization, continuity and irregularities have been recorded into the knowledge base. If yes, use these to adjust decomposition parameters. If not, compute the parametrization factors and set the proper tolerance and jitter factors as described above.
3. Decompose the NURBS into smaller pieces using the tolerance and jitter factors.

4. Bin the segments into a spatial data structure for fast nearest neighbor search.
5. For all query points use the wave model to find nearest segments.
6. Refine the nearest segment, with a somewhat finer subdivision, to get a start point for fast Newton iteration.
7. Call Newton with the guess parameter and account for anomalies such as cusps and boundary points.

The steps that require additional details are the spatial data structure setup and the wave model based start point search. Spatial data structures for nearest neighbor search have been the subject of extensive research. Lots of models have been set up ranging from simple binary partitioning trees through Voronoi-based methods to fancy structures like *vp-trees* (vantage point tree). Our method is based on a simple uniform grid subdivision primarily because of the following:

- NURBS entities are subdivided into nearly equal segments that naturally call for a uniform space division.
- Implementing the wave model requires a grid structure and although the size of the grid in each direction could be different, for ease of computation uniformity is enforced.
- More complicated techniques such as octrees or Voronoi diagrams require quite a bit of computational cost, the majority of which is memory management. Dynamic memory allocation and reallocation can be more costly than floating point operation, and significantly more costly than running empty loops (empty cells cause the algorithm to jump, i.e. the loop will run without doing any computation).

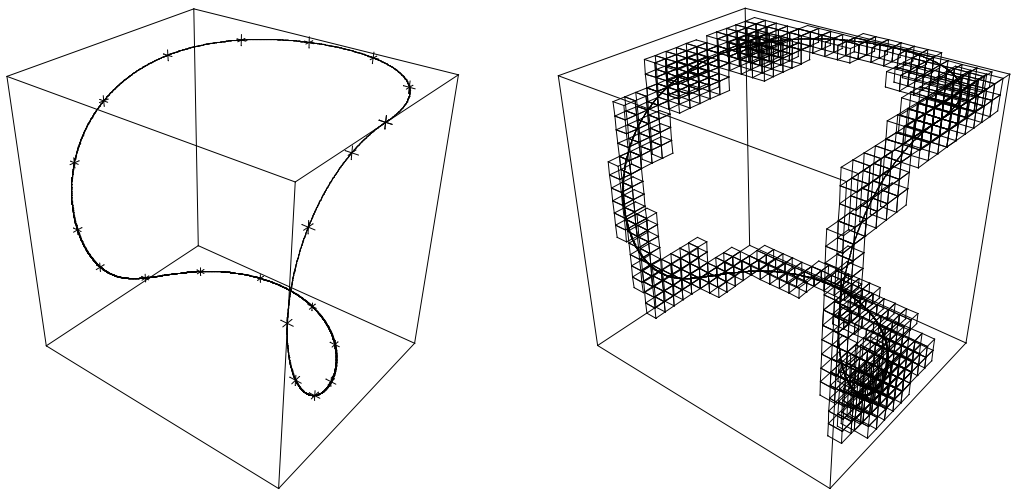


Fig. 5: Curve example with min-max box: decomposition (left) and registering segments with cells (right).

The major steps of data binning are as follows:

1. Compute the min-max box of the NURBS. Note that refinement is needed in order to avoid large bounding boxes. Some operations, such as interpolation, may produce control points that are way to far from the NURBS curve or surface.
2. Compute the extents of the box in x-, y- and z-directions and get the maximum of these. The size of the spatial cell is the maximum extend divided by the number of subdivision points for curves, or the maximum subdivision points for surfaces in u- or v-directions.
3. Now augment the bounding box by adding the query points and compute the x-, y- and z-resolutions by dividing the updated x-y-z extents with the cell size. The reason why the cell size is computed based on the NURBS' bounding box is because the query points can lie quite far from the NURBS giving rise to large cell sizes which would nullify the localization provided by the subdivision.
4. For each cell provide a buffer of fixed size (our implementation used 15). This avoids constant memory reallocation which would slow down the program. Update the buffers if they happen to overflow during data binning (on average each buffer contains only a few entity references).
5. Using the subdivision points (parameters) extract NURBS sub-curves or sub-patches and bin them into the cell data structure.

6. For each NURBS segment, find the cells that contain this segment. This is done by computing the min-max box of the segment and getting the indexes of the cells that contain the entire min-max box. To avoid multiple registrations, check the cell if it contains the segment index already.

This process, although quite a bit of programming, is very cheap as the most expensive part is bounding box computation. Memory is allocated once for the multidimensional index array that holds the segment indexes and reallocation is nearly never done. After the data binning is completed, each small NURBS segment or patch is registered with one or more cells in the uniform data structure. Fig. 5 shows an example of data binning. On the left the curve, embedded into its bounding box, is decomposed into nearly equal segments using $tol = 0.5$ and $jit = 80\%$. The right side of the figure shows all the cells the segments are registered with. Note that some cells do not intersect the curve. This is because the individual pieces are binned into the structure using their bounding boxes. Using other techniques, such as a fine control polygon obtained via knot refinement, could produce a cell structure with less cells. However, the extra cost is not worth the effort because query points may lie far from the curve requiring cells, not intersecting the curve, to be visited. The algorithm below applies proper book keeping to make sure that cells are visited no more than once and to avoid multiple sampling of curves/surfaces as well.

After the segments/patches have been binned, the algorithm is ready to send in the waves to find closest points. The critical steps are as follows:

1. Mark each cell as non-visited and each segment as non-sampled. This bookkeeping avoids multiple visits to cells and multiple sampling of segments.
2. Find the cell the query point is in, and initialize the search distance with the shortest distance between the query point and the closest plane of the cell.
3. While not done do
   - 3.1 Propagate the wave from the query point and visit cells surrounding the unvisited cells.
   - 3.2 If a cell is found to be un-visited, find the segment registered with the cell.
   - 3.3 If the segment is not yet sampled, obtain a nearly uniform sampling using the same method described above. Note that the object gets decomposed two times: once the entire object gets decomposed to obtain a coarse subdivision for segment binning, and the candidate segments get sampled again, possibly with a finer subdivision, for finding start points/parameters.
   - 3.4 Obtain the nearest point and the corresponding parameter and update the minimum distance.
   - 3.5 Mark the cell visited and the segment sampled.
   - 3.6 If the minimum distance is less than the search distance, done.
   - 3.7 Update the search distance by the cell size.
4. Perform Newton iteration with the given start parameter(s).

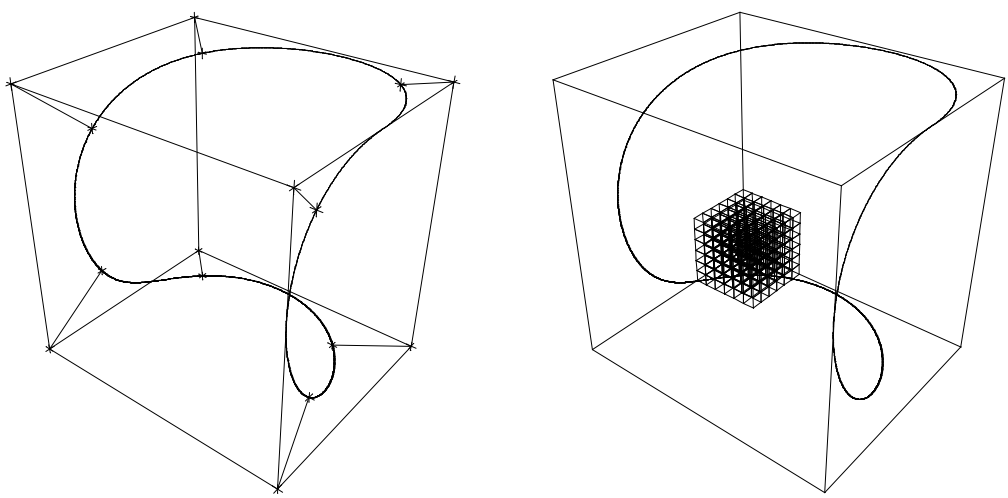


Fig. 6: Shortest distances of query points (left) and the extent of wave penetration (right).

This process is then repeated for each query point without having to change the spatial data structure or sampling segments that have been sampled already. Simply put, for each query point a cubical wave is sent towards the object and it is allowed to penetrate the object until it is found that the shortest distance is smaller than the distance to the boundary of the expanding wave. Fig. 6 shows an example. On the left part shortest distances from the corners of the bounding box are computed, whereas the right figure shows how far the cubical wave penetrates into the spatial subdivision. Note that only two candidate segments have been found and cubes that do not have segments inside are simply skipped over.

7. TESTS AND EXAMPLES

The algorithm above has been tested on numerous examples, both for curves and surfaces. Two fairly difficult surface examples are shown in Fig. 7 below. Both the dust pan, Fig. 7 left, and the brush handle, Fig. 7 right, contain high as well as low (zero) curvature areas, small neck areas and wide planar parts, and parts where curvature changes rapidly from zero to a high value. The dust pan is used to test how the algorithm handles mostly concave parts, whereas the brush handle provides a convex skin from which distances are computed. In addition, both of them contain plenty of boundary cases, i.e. the closest point falls on the boundary.

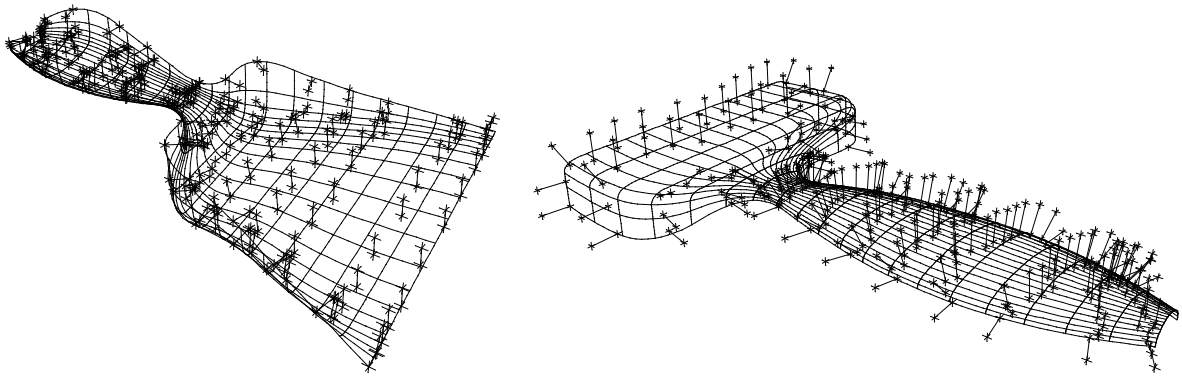


Fig. 7: Distances of random points from dust pan (left) and brush handle (right).

Numerical tests have been performed on a large set of curves and surfaces. The data provided below are for the curve in Fig. 6 and for the two surfaces in Fig. 7. In all cases the point coincidence tolerance was set to 10^{-6} and the perpendicular tolerance was 10^{-4}. One hundred (100) random points were used to compute distances and to study the behavior of the algorithm. The decomposition as well as the sampling tolerances were changed with a constant jitter of 80%. The questions to which we wanted to find answers were: (1) how well Newton behaved, i.e. how long it took to find the solutions, (2) what was the optimal spatial subdivision, and (3) how far we could push the method to get real-time results. The tables below are representatives of consistent results across a wide variety of NURBS entities.

ε_D	10%	10%	10%	5%	**5%**	**5%**	1%	1%	1%
ε_S	10%	5%	1%	10%	**5%**	**1%**	10%	5%	1%
nit	1><2	1><2	1	1><2	≈1	<1	<1	<1	<1

Tab. 2: Convergence result for the curve in Fig. 6.

The notation is interpreted as follows: ε_D is the decomposition relative tolerance, e.g. for 10%, the data structure in the worst case is a $10\times10\times10$ grid, ε_S is the sampling tolerance, so for 5% roughly 20 points are sampled nearly uniformly, nit denotes the number of iterations; 1><2 implies that the number of iterations is one or two, ≈1 means that convergence is reached in 1 iterations almost always, <1 means that Newton needed zero or 1 iteration to find the closest point, and the column with bold numbers indicate the recommended optimal setup. The result clearly illustrates that the key to proper distance computation via Newton is proper data preparation.

Let us now look at the numerical indicators for the dust pan and the brush handle. Tables 3 and 4 below contain the results. It is worth noting right here that Newton has a much more difficult time with surfaces than with curves, i.e. converging to the solutions takes much longer for surfaces than for curves (two equations must be satisfied simultaneously and two new parameters must be computed so that the candidate point moves towards the foot-point of the perpendicular). Therefore finding good start points it more critical for surfaces than for curves.

$\varepsilon_D^u, \varepsilon_D^v$	10%	10%	**10%**	5%	5%	5%	1%	1%	1%
$\varepsilon_S^u, \varepsilon_S^v$	10%	5%	**1%**	10%	5%	1%	10%	5%	1%
nit	1><2	1><2	1	1><2	≈1	1	1	<1	<1

Tab. 3: Convergence result for the dust pan.

$\varepsilon_D^u, \varepsilon_D^v$	10%	10%	**10%**	5%	5%	5%	1%	1%	1%
$\varepsilon_S^u, \varepsilon_S^v$	10%	5%	**1%**	10%	5%	1%	10%	5%	1%
nit	1><2	1><2	1	1><2	≈1	<1	<1	<1	<1

Tab. 4: Convergence result for the brush handle.

The tables show and extensive testing clearly indicate that proper curve/surface subdivision and spatial data structure setup allows Newton to always converge within one or two iterations. Because it is faster to compute points on NURBS entities than managing memory or computing higher derivatives, the following is recommended:

- Set up the spatial data structure based on a coarse subdivision. For curves 5% is recommended and surfaces can make do just fine with 10%.
- Sample the sub-segments at 1% density to get very good start values. This allows Newton to converge in one iteration on average. Since the sampling is done only once, computing distances for multiple points creates practically no overhead.

It is simply not worth the effort to set up a very fine spatial data structure, and in fact, as the level of subdivision increases, the amount of memory grows exponentially, i.e. a 1% surface decomposition may result in 1 million cells!

The question of how fast the algorithm is depends on the computing environment, both hardware and software. The method uses a large array of capabilities from KGnurbs™ all of which are fine tuned, i.e. reproducing the timing results is nearly impossible. However, just to provide a flavor, we were able to compute distances of random points for curves up to 10,000 in real time using an off-the-shelf laptop, using a 1.73GHz processor and 2GB RAM, with no optimization or other kind of fancy gadgets. Surface timings were much slower; roughly 1,000 points were computed in real time for the complicated surface types in Fig. 7.

8. CONCLUSIONS

Newton type methods have been hounding the CAD community for decades. While the underlying math is very simple, distance routines have been tweaked to accommodate individual failures. This paper presented a knowledge-guided approach to a stubborn problem: (1) identify critical object types and apply non-numerical techniques, (2) use knowledge about parametrization, continuity and irregularities to adjust the algorithm and to make intelligent decisions in case Newton is unable to move fast enough toward the solutions, and (3) set up a spatial data structure for fast nearest neighbor search based on a discrete version of the wave model, and to find appropriate start parameters guaranteeing convergence within one or two iterations. The method works well with multiple points distributed randomly and generalizes to other query object types, such as lines, planes and curves.

9. ACKNOWLEDGEMENTS

This research was supported by the National Science Foundation under grant DMI-0758231, awarded to the University of South Florida. All opinions, findings, conclusions and recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the National Science Foundation or the University of South Florida.

10. REFERENCES

[1] Besl, P. J.; McKay, N. D.: A method for registration of 3-D shapes, IEEE Transaction on PAMI, 14, 1992, 239-256.

[2] Chen, X.-D.; Yong, J.-H.; Zheng, G.-Q.; Paul, J.-C.; Sun, J.-G.: Computing minimum distance between two implicit algebraic surfaces, Computer-Aided Design, 38, 2006, 1053-1061.

[3] Dyllong, E.; Luther, W.: Distance calculation between a point and a NURBS surface, in Laurent, P. J., Schumaker, L. (Eds.), Curve and Surface Design, 1999, Saint-Malo, 55-62.

[4] Gilbert, E. G.; Foo, C.-P.: Computing the distance between general convex objects in 3-D, IEEE Transaction on Robotics and Automation, 6(1), 1990, 53-61.

[5] Gilbert, E. G.; Johnson, D. W.; Keerthi, S. S.: Fast procedure for computing the distance between complex objects in 3-D, IEEE Journal of Robotics and Automation, 4(2), 1998, 193-203.

[6] Jiminez, P; Thomas, F.; Torras, C.: 3-D collision detection: a survey, Computers and Graphics, 25(2), 2001, 269-285.

[7] Henshaw, W. D.: An algorithm for projecting points onto a patched CAD model, Engineering with Computers, 18, 2002, 265-273.

[8] Hu, S.-M.; Wallner, J.: A second order algorithm for orthogonal projection onto curves and surfaces, Computer-Aided Geometric Design, 22, 2005, 251-260.

[9] Johnson, D. E.; Cohen, E.: A framework for efficient minimum distance calculations, Proc. IEEE International Conference on Robotics and Automation, 1998, 3678-3684.

[10] Kim, K.-J.; Minimum distance between a canal surface and a simple surface, Computer-Aided Design, 35, 2003, 871-879.

[11] Larsen, E.; Gottschalk, S.; Lin, M. C.; Manocha, D.: Fast distance queries with rectangular swept sphere volumes, Proc. EEE International Conference on Robotics and Automation, 2000, 3719-3726.

[12] Lee, K.; Seong, J.-K.; Kim, K.-J.; Hong, S. J.: Minimum distance between two sphere-swept surfaces, Computer-Aided Design, 39, 2007, 452-459.

[13] Lennerz, C.; Schoemer, E.: Efficient distance computation for quadric curves and surfaces, Proc. Geometric Modeling and Processing, 2002, 60-9.

[14] Lin, M. C.; Canny, J. F.: A fast algorithm for incremental distance calculation, Proc. IEEE International Conference on Robotics and Automation, 1991, 1008-1014.

[15] Lin, M. C.; Gottschalk, S.: Collision detection between geometric models: a survey, Proc. IMA Conference on Mathematics of Surfaces VIII, 1998, 37-56.

[16] Ma, Y. L.; Hewitt, W. T.: Point inversion and projection for NURBS curve and surface: control polygon approach, Computer-Aided Geometric Design, 20, 2003, 79-99.

[17] Piegl, L. A.: Knowledge-guided NURBS: principles and architecture, Computer-Aided Design and Applications, 3(6), 2006, 719-729.

[18] Piegl, L. A.; Tiller, W.: The NURBS Book, Springer-Verlag, Second Edition, 1997.

[19] Piegl, L. A.; Tiller, W.: Symbolic operators for NURBS, Computer-Aided Design, 29, 1997, 361-368.

[20] Ristic, M.; Brujic, D.: Efficient registration of NURBS geometry, Image and Vision Computing, 15, 1997, 925-935.

[21] Selimovic, I.: Improved algorithms for the projection of points on NURBS curves and surfaces, Computer-Aided Geometric Design, 23, 2006, 439-445.

[22] Sohn, K.-A.; Juettler, B.; Kim, M.-S.; Wang, W.: Computing distances between surfaces using line geometry, Prof. Pacific Graphics, 2002, 236-245.

[23] Thomas, F.; Turnbull, C.; Ros, L.: Cameron, S.: Computing signed distances between free-form objects, Proc. IEEE Conference on Robotics and Automation, 2000, 3713-3718.

[24] Turnbull, C.; Cameron, S.: Computing distances between NURBS-defined convex objects, Proc. International Conference on Robotics and Automation, 1998, 3686-3690.