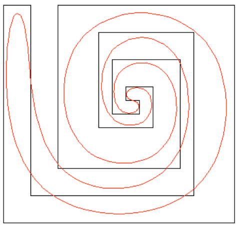
Education-Driven Research in CAD



Jarek Rossignac

College of Computing and GVU Center

Georgia Tech, Atlanta, USA

ABSTRACT

We argue for a new research category, which we call Education-Driven Research (abbreviated EDR), which fills the gap between traditional Research in any specific field (such as CAD), which is concerned with educational objectives, and research in Education, which is focus on fundamental teaching and learning principles and possibly their customization to broad areas, such as mathematics or physics. The objective of EDR is to simplify the formulation of the underlying theoretical foundations and of specific tools and solutions, so as to make them easy to understand and internalize. As such, EDR is a difficult and genuine research activity, which requires a deep understanding of the specific field and usually cannot be carried out by generalists with primary expertise in broad education principles. We illustrate the value of EDR with three examples in CAD: (1) the Split&Tweak subdivisions of a polygon and its use for generating curves, surfaces, and animations; (2) the construction of a topological partition of a plane induced by an arbitrary arrangement of edges; and (3) a romantic definition of the minimal and Hausdorff distances. These examples demonstrate the value of using analogies, of introducing evocative terminology, and of synthesizing the simplest fundamental building blocks. The intuitive understanding provided by EDR enables the students (and even the instructor) to better appreciate the limitations of a particular solution and to explore alternatives. In particular, un these examples, EDR has allowed the author to: (1) reduce the cost of evaluating a cubic B-spline curve; (2) develop a new curve that approximates a control polygon better than either a cubic B-spline or an interpolating 4-point subdivision curve; (3) discover how a circuit inclusion tree may be used for identifying the faces in an arrangement; and (4) rectify a common misconception about the computation of the Hausdorff error between triangle meshes. We invite the scientific community to encourage the development of EDR by publishing its results as genuine Research contributions in peer-reviewed professional journals.

***Keywords**: Education-Driven Research, B-spline Curves, Polygon Subdivision, Four-point Subdivision, Parametric Surfaces, Evaluation of Curves and Surfaces, Plane Arrangements, Point-Set Topology, Polygons, Faces, Circuit Inclusion Tree, Non-Manifold Modeling; Minimum Distance, Hausdorff Distance.*

INTRODUCTION

As a new educator in the young and rapidly evolving areas of Computer Aided Design and 3D Graphics, I found myself initially devoting the majority of my efforts to the selection of lecture material that would endow student with what I believed to be the essential foundations and to the organization of this material into a logical and well paced progression. I quickly discovered that this careful preparation would not bear fruits, unless complemented by an engaging delivery that kept students attentive and motivated. I realized that I could best engage the students by inviting them, during my lectures, to discover why a proposed solution or formulation was wrong and to invent a better one. Although students usually like this style of lecture, most struggle when they lack simple intuitive models of the concepts that they have to manipulate during such a creative exercise. I attribute this deficiency to the discrepancy between the elegant formulations promoted in scientific publications and the intuitive, often much simpler, mental models that are helpful when probing the validity of a solution, looking for counterexamples, or inventing proofs. Hence, I started devoting a large fraction of my educational effort to the exploration of such simplified and intuitive interpretations of the technical material I teach. I propose to use the term Education-Driven Research (abbreviated EDR) to describe such an activity. Note that EDR is difficult because synthesizing a new and simpler formulation requires an in-depth understanding of the technical field and of the true essence of the concepts or algorithms being explored. In a sense, it is the culmination of Research.

I consider EDR as an integral and possibly the most valuable part of my research, because it empowers my students, and myself, to make better use of prior research results. I am convinced that many of my colleagues do the same. Unfortunately, we do not have a formal venue for disseminating our results in EDR. One will rightly argue that textbooks are a good place to publish EDR results. Unfortunately, many faculty may have discovered precious EDR formulations for specific topics, which they would be happy to share with colleagues, but do not have the time, or courage, to write a complete textbook. Although numerous EDR results are available on the Internet, they have not benefited from a formal review and selection process and are often lost amongst uninspired attempts. Finally, considering other priorities, a faculty member may put off the effort of producing a carefully constructed EDR paper forever, unless it will count as a peer-reviewed research contribution. For all these reason, I believe that each scientific community must welcome the publication of EDR results in professional journals and should evaluate them as genuine Research contributions.

The remainder of the paper presents three examples of EDR which I have developed for my Foundations of 3D Graphics class: (1) a polygon subdivision rule for generating curves and surfaces, (2) an approach for computing the faces of a partition of a plane by an unstructured arrangement of edges, and (3) an intuitive formulation of Hausdorff distance and its use to check the validity of algorithms that compute it. We conclude by a discussion of the EDR principles used in these examples and of their benefits and limitations.

POLYGON SUBDIVISION

In spite of my considerable efforts to clarify the various formulations and constructions of B-spline and Bezier curves and surfaces, informal polls have revealed that this was the least popular subject amongst the students in my graduate class in Computer Graphics. When faced with the challenge of teaching the subject at an undergraduate level, I concluded that the elegant formulations commonly used to present B-spline and Bezier curves and surfaces [Fari93] hide the simplicity with which their generation can be implemented. I decided to forgo generality and to focus on simple subdivision processes that generate uniform cubic B-splines and 4-point subdivision curves and on their use for producing static and animated free-form surfaces. Because of the simplicity of these subdivision rules, I can require that the students learn them by heart and implement them in the first weeks of class. Furthermore, the intuitive formulation of these subdivision rules enables the students to invent their own variations and to explore numerous applications.

Initially, I consider a closed loop polygon. It may, but needs not, be planar. We refine it by the following "**Split&Tweak**" process. First, **split** each edge by inserting a **new** vertex in the middle of it. Then **tweak** either the new vertices, or the old ones, or both. Repeat the process as desired. After each Split&Tweak **refinement**, the number of vertices in the polygon has doubled. A good tweak rule will increase the smoothness of the polygon with each refinement.

After this short introduction, students are invited to try and invent their own tweak rules. Then, I offer two, whose formulation is based on averaging points and on finding (first degree) neighbors and second degree neighbors (i.e. neighbors of neighbors) along the polygon. A tweak step first computes a displacement vector for each vertex from the locations of its neighbors and then applies these displacements.

B-spline tweak

This is a surprisingly little known subdivision scheme for cubic B-splines (Fig. 1).

B-spline tweak*: Move the **old** vertices half-way **towards** the **average** of their new **neighbors.***

Repeated refinements that use a B-spline tweak produce a polygon that converges to a **uniform cubic B-spline** curve, which has a piecewise polynomial formulation. The resulting curve is usually very smooth but undercuts the corners and does not interpolate the initial vertices, which are called the **control** points of the B-spline curve.

Note that these split and tweak operators may be implemented as linear combinations of mid(X,Y), which returns the midpoint (X+Y)/2 and can thus be implemented using a single add-and-shift per coordinate. In 3D, each Split stage may be performed as 3 additions and 3 one-bit shifts per vertex. Each Tweak requires 6 additions and 6 shifts. For example, K=mid(A,B) and B'=mid(B,mid(L,K)). Consider that we have n vertices as the result of the penultimate iteration. The last Split&Tweak step will perform 9n add-and-shift operations and will create n new vertices. Thus, the total number of add-and-shift operations is 9(n+n/2+n/4+…), which sums up to less than 18n. Because the final polyline has 2n vertices, the total cost per final vertex is 9 add-and-shift operations, which represents a saving, when compared to previously published alternatives [BaGo88].

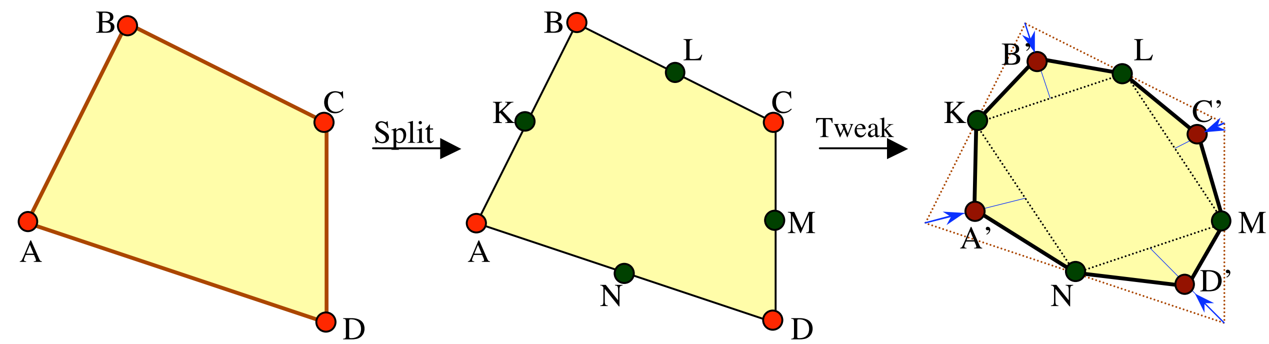


Figure 1: The closed loop control polygon with vertices A, B, C, and D (left) is subdivided in 2 steps. First (center), the "split" step inserts new vertices (K, L, M, and N) in the middle of each edge. Then (right), the "B-spline tweak" step adjust the original vertices (A, B, C, and D) by moving them half-way, towards the average of their new neighbors. The adjusted vertices are labeled A', B', C' and D'.

4-point tweak

This is the standard **4-point subdivision** scheme (Fig. 2).

4-point tweak: *Move the **new** vertices by one quarter **away from** the average of their **second-degree neighbors**.*

The resulting curve interpolates the initial vertices and bulges out through the edges.

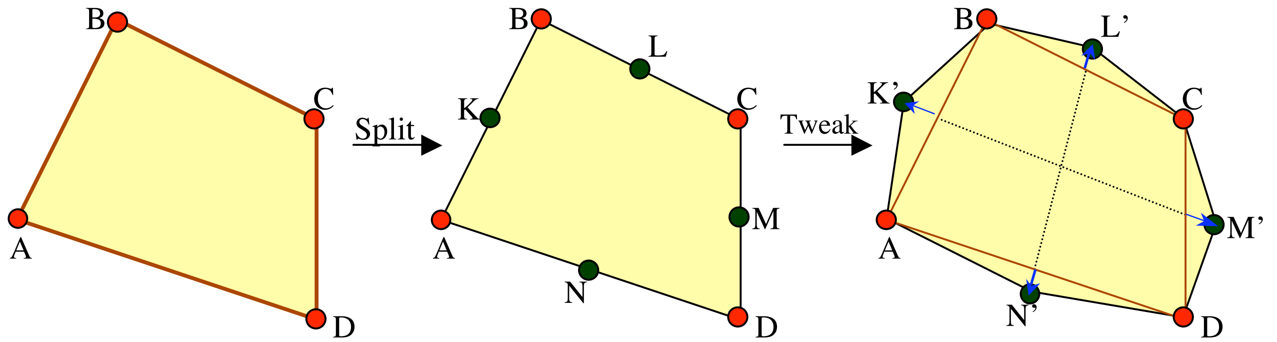


Figure 2: The closed loop control polygon with vertices A, B, C, and D (left) is subdivided in 2 steps. First (center), the "split" step inserts new vertices (K, L, M, and N) in the middle of each edge. Then (right), the "4-point tweak" step adjust the new vertices (K, L, M, and N) by moving them by one quarter away from the average of their second-degree neighbors. The adjusted new vertices are labeled K', L', M' and N'. For example, the adjustment vector LL'=(L–(K+M)/2)/4. Note that equivalently, we can use LL'=(L–(A+D)/2)8.

Jarek's tweak

The simplicity of this Split&Tweak formulation has revealed that, when the interpolation of the control points or a piecewise polynomial formulation are not required, a compromise between these two tweak rules may be preferred to either one of them (Fig. 3).

Jarek's tweak: *Move the old vertices by half of the B-spline tweak and the new vertices by half of the 4-point tweak displacements.*

The three schemes are illustrated in Fig. 4, which shows that Jarek's refinement produces a curve that lies in-between the B-spline and the 4-point curves and is a closer approximation to the original polygon than either of these two.

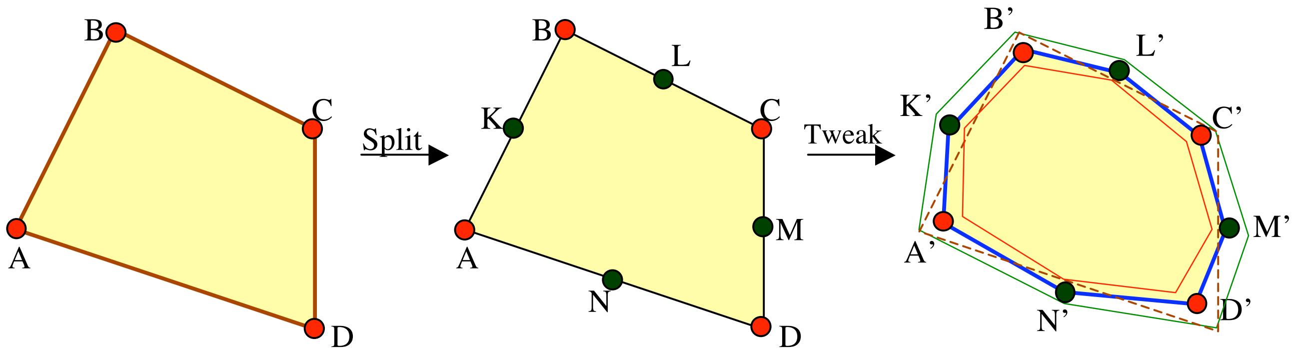


Figure 3: The closed loop control polygon with vertices A, B, C, and D (left) is subdivided in 2 steps. First (center), the "split" step inserts new vertices (K, L, M, and N) in the middle of each edge. Then (right), "Jarek's tweak" adjust the old vertices by half the displacements suggested by the B-spline tweak and the new vertices by half the displacement suggested by the 4-point tweak. Note that for convex control polygons, Jarek's refined polygon (blue) lies between the results of the Bspline (red) and 4-point (green) refinements.

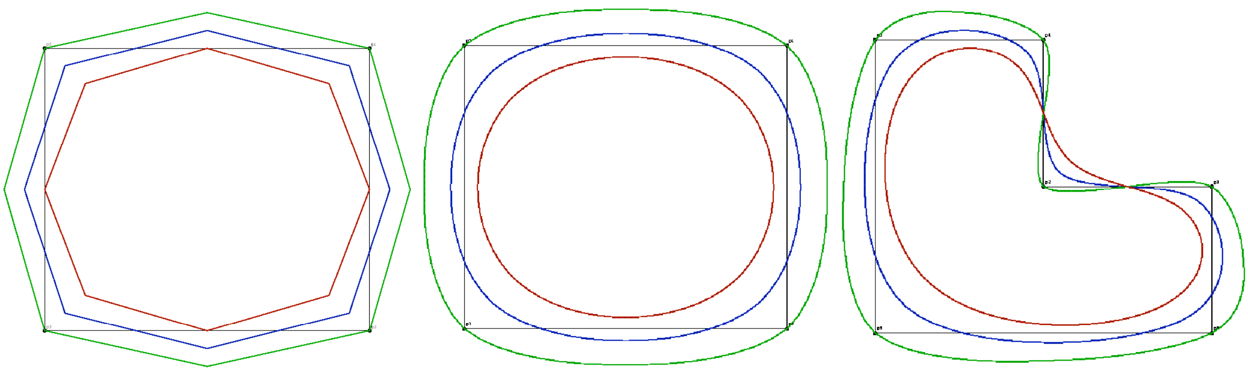


Figure 4: The original control polygon is a square (black). The results of the first Split&Tweak iteration are shown left: the 4-point tweak is shown in green, the B-spline tweak is shown in red, and jarek's tweak is shown in blue. The corresponding final curves are shown for the same initial control polygon (center) and for an L-shaped control polygon (right).

Although simple variations of Split&Tweak exist for converting a B-spline formulation into a Bezier form and for further refining the Bezier segments, I refrain from discussing them at this point. I also deliberately omit the discussion of the analytical formulation of the curves and of the convexity properties of the B-spline scheme. These will be better motivated and understood when geometric intersections are studied.

Open control polygons

For simplicity, we have defined Split&Tweak for close-loop polygons. Notice that each edge of the original polygon is refined into a chain of edges that we call a **segment** of the final curve. To Split&Tweak an open polygon that has a first and a last vertex, first close the polygon by adding an edge joining the last and the first vertex; then perform the Split&Tweak steps as before; finally remove 3 segments: the one corresponding to the edge you have added and its two neighbor segments.

Animation

Let t denote the index of vertex C[t] along a refined curve C. If, for example, we start with a closed curve of 5 control points and perform Split&Tweak 3 times, we obtain 5x2x2x2=40 vertices: C[0], C[1], … C[39].

By changing the value of t, we control the motion of a point along the refined curve. The motion steps are discrete, the point jumps from a vertex to the next one. If we need more resolution, we can subdivide the curve further. (we could also slide along the edges for a continuous motion, but we will ignore this option here.) Thus, the initial polygon and the Split&Tweak process could be used to define the motion of a point in 2D or 3D and also to control the evolution of the various parameters of an animation. We can think of t as the time parameter in this context.

Surfaces

We can think of a surface as the set swept by a refined curve, S(v), as the parameter v evolves in some chosen interval. To do this using our Split&Tweak tool, we consider an animated control polygon whose control points are moving, each along a different curve. Their motions are synchronized using the parameter v. Suppose for example that we have 4 such control points. The first one moves along curve A. Its position for any given value of v is defined by A[v]. The other vertices move similarly along curves B, C, and D. To get an approximation of this surface, we use Split&Tweak to refine A, B, C, and D to the desired accuracy, as explained above. Assume for simplicity that their original polygons have all the same number of vertices and that we perform the same number of Split&Tweak refinement steps on each curve. (We may easily remove these restrictions by parameterizing each curve from 0 to n–1. Also note that we do not have to use the same tweak rule for each one of these curves.) Suppose that in the end, each curve has n vertices. Then, for each value of v between 0 and n–1, we consider the polygon with vertices (control points) A[v], B[v], C[v], and D[v] and refine it by a series of Split&Tweak operations. We obtain a refined polygon with vertices S(v)[0], S(v)[1], … S(v)[m–1]. We can save this row of m vertices for each value of v and get an array M[v,u] of mxn vertices. We can render the corresponding surface as quads (M[v,u], M[v+1,u], M[v+1,u+1], M[v,u+1]).

Note that we can use a different tweak rule for S than for the curves A, B, C, and D. Furthermore, we can treat all of them as closed loop curves, in which case we get a torus. Alternatively, we can treat S(v) as a closed-loop curve, but treat A, B, C, and D as open curve segments. In this case, we get a cylinder. Finally, if all five formulations are open, we get a rectangular patch. When a B-spline tweak is used for all curves, the resulting quad-mesh converges to the standard bi-cubic uniform B-spline surface. Examples produced by the students are shown Fig. 5. The interface of an online Split&Tweak editor written in Java by an undergraduate student is shown in Fig. 6.

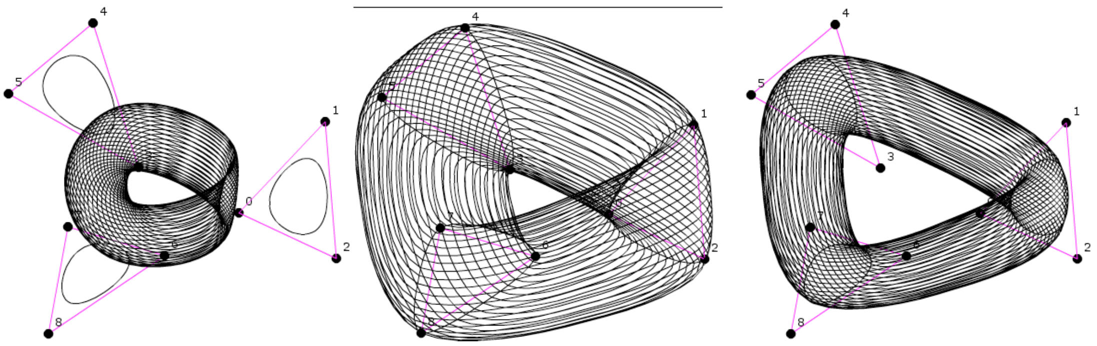


Figure 5*: Surfaces generated from the same 3 control triangles through a Split&Tweak process. B-spline Tweak rules were used for all curves (left). 4-point tweak rules were used for all curves (center). B-spline tweaks were used for the three control curves and a 4-point tweak rule was used for the swept curve (right).*

Animated surfaces

Finally, to produce an animated surface, define each control point of polygons A, B, C, and D as a moving point whose motion along an animation curve is parameterized by a time variable t. Specify each one of these animation curves in terms of an initial control polygon and a tweak rule. Refine them to the desired accuracy using one of our Split&Tweak procedures. *Et voilà,* you have an animated surface controlled by a “few” control points.

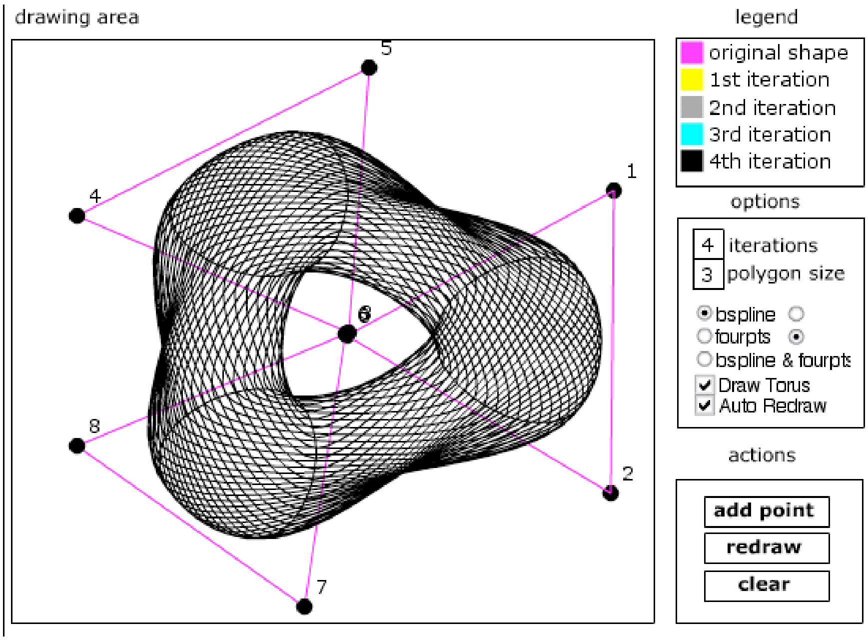


***Figure 6**: The Java-based web editor written by undergraduate student Jessie Shieh for my class project on the Split&Tweak refinements lets users specify the control polygons, select one of the three Tweak rules, and specify whether the curves are closed or not. Here, it shows genus-one surface defined by 9 control points.*

TOPOLOGY OF AN ARRANGEMENT

Arrangements of lines in the plane have been studied in Computational Geometry texts [deBe&97]. Many intricate algorithms in CAD/CAM require that we identify the faces of a plane that are defined by an arrangement of edges. One difficulty of this task lies in the fact that we do not share a common definition of the most primitive topological terms, such as **face** or **polygon**. In fact, I often ask my students (and colleagues) at the beginning of a lecture to each write down the precise definition of a polygon. As you can imagine, the results are amusing, sometimes alarming, since even experts will produce invalid definitions or define a polygon in terms of a data structure, such as a linked list of vertices. Then, I draw a few shapes on the board and ask them to agree which ones are polygons and then to fix their definition to be consistent with the choices. To define polygons properly, I like to first introduce a few topological terms, at a rather intuitive level. I call that my “1-2-3 Topology clinic” I will confine my discussion here to set in 2D.

Topology

Consider a set S and its complement, denoted $\underline{S}$. Often, we represent S (and hence $\underline{S}$) by its boundary, bS. The boundary is the set of points that are adjacent to both S and to $\underline{S}$. By this, we mean that a point p is in bS if any ball of a strictly positive radius around p, no matter how small, contains points of S and of $\underline{S}$. We use the standard set theoretical Boolean operators: A–B is the set of points of A that are not in B and AB is the set of points where A and B overlap. We can now decompose the plane into 6 mutually disjoint sets (Fig. 7):

- The **interior** iS of S, defined as S–bS, is the set of points of S that are not on its boundary.
- The **exterior** eS of S, defined as $\underline{S}$–bS, is the set of points out of S that are not on its boundary.
- The **skin** sS of S, defined as biSS using a compact notation for (b(is))S, is the portion of the boundary of S that is included in the set S and is separating S from its complement.
- The **wounds** wS of S, defined as beS$\underline{S}$, is the portion of the boundary that is not included in the set S and separates S from $\underline{S}$. Wounds may be thought of as the places where the skin is missing.
- The **hair** hS of S, defined as beSS, is the set of isolated points and dangling edges of S that are not separating S from $\underline{S}$. The hair is adjacent to eS, but not to iS. The hair sticks out.
- The **cuts** cS of S, defined as biS$\underline{S}$, are the lower-dimensional holes (isolated vertices and edge-cracks) in iS. The cut is adjacent to iS, but not to eS.

A set S is **open** if it equals its interior, i.e. it has no skin or hair. A set is **closed** if it has no wound or cut. The **closure**, kS, of a set S is the union iSbS, or equivalently ScSwS. A set S is “**clean**”, when it has no wound, no hair, and no cut. Such sets are said to be “**regularized’** in the Solid Modeling literature. To convert an arbitrary set S into a clean set, we need to cut its hair, mend its cuts, and grow back the skin over its wounds. The regularized (“clean”) version rS of S may be formally defined by iScSsSwS.The effect of these various operators in shown in Fig. 7.

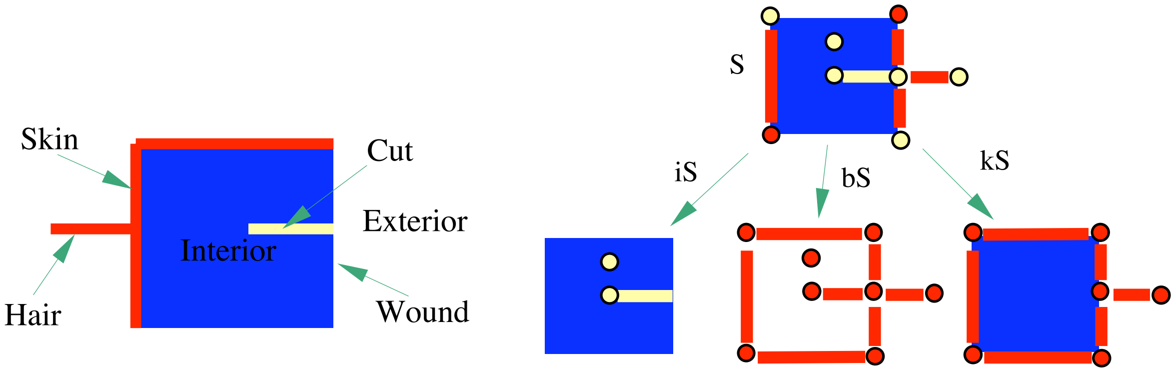


Figure 7: The interior, exterior, and the four boundary types are illustrated (left). The effect of the interior, boundary, and closure operators are illustrated on a non-regularized set (right).

Polygons

Several definitions of polygon make sense. I define polygons as regularized subsets of the plane whose boundary is contained in a finite union of lines. Furthermore, I restrict polygons to be connected and bounded (i.e. not extending to infinity).

Arrangement of edges in the plane

Now consider a random set of edges (line segments) on the plane. If you subtract their union from the plane, you get a set of connected components, that we will call **faces**. Note that faces are not polygons. For one, they are open. They may also be unbounded and may have cuts. We want an algorithm for identifying the faces of the arrangement and for building a representation that explicitly tells us which faces are adjacent (share bounding edges) and which are holes in other faces. To appreciate the value of the metaphor described below, I invite the reader to pause here and develop an algorithm for solving this problem before reading the rest of this section.

The sidewalk circuit metaphor

Here's the metaphor I use in class: Edges are streets. On each side of each edge, and around dead-ends, there is a sidewalk. Note that the **sidewalks** never cross the streets. Now, pick a child, give her a brush and a bucket of paint, put her on a clean side-walk and say "Walk and paint a line behind you until you get back here. Stay on the sidewalk. Never cross any street!" Note that they always come back to their starting point. Once she is back, put another kid on a clean (unpainted) sidewalk with a bucket of paint of a new color and send him around. Keep doing this until all sidewalks are colored. We say that each child traces a **circuit**.

Inclusion tree

Now, we need to establish an **order** for the circuits. We will store this partial order in an **inclusion tree**. We assume that there is a infinite circuit around the universe. This will be the root of our inclusion tree. Then, we pick an arbitrary circuit and make it a child of the root. We consider the other circuits one by one and insert them where they belong in the tree. The insertion process puts a candidate circuit-node as a new root of the tree and then moves it down the tree, at each step making it the child of the circuit-node that "contains" it. When no such circuit-node may be found, we keep the candidate circuit-node there and keep as its children the circuit-nodes contained in the candidate and promote its other children to be its siblings. We to end up with a tree where each circuit-node is contained in its parent.

What does it mean for a circuit-node to **contain** another one? The circuit-node of the infinite circuit contains, by definition, all other circuit-nodes. A node representing circuit A is said to contain a node representing circuit B if B lies inside the finite face bounded by A. To test this, cast a ray from a point on B to infinity and count the number of times the ray intersects A. If that number is odd, then A contains B. In the context of our metaphor, a circuit B is inside a circuit A if you could not walk to infinity from a point on B without crossing A. The whole approach is illustrated in Fig. 8.

Faces of an arrangement

Once the tree is built, the **faces** are defined by their outer circuit, which always corresponds to a node that can be reached from the root by traversing an odd number of edges, and by the circuits of its child-nodes that identify the boundaries of the holes in the face.

Discussion

To be honest, the simplicity of this solution is hiding the most delicate aspect of its implementation, namely the accuracy problems in computing the sidewalks, which assume that you have identified all of the intersections between edges and correctly sorted them along each edge. This may be a good opportunity for a discussion of numeric robustness in geometric computation. If not, we can ask the students to implement this solution in a safe digital environment where all of the edges are axis aligned, all of the coordinates are integer multiples of 5. This precaution makes all intersection calculations error-free and ensures that if you start from the middle of a sidewalk segment and walk in a direction orthogonal to the sidewalk, you will never run into a vertex of an edge or into the intersection between two edges. In fact, with these precautions, the algorithm could be performed on a discrete pixel grid where the sidewalks could be color-coded and would be formed of pixels that always have two edge-connected neighbors of their color.

The metaphor of sidewalks and the topology terminology introduced above come in handy when discussing algorithms for further analysis of the arrangement. For example, we may want to convert faces into polygons, by regularizing them, which involves removing their cuts and identifying their wounds.

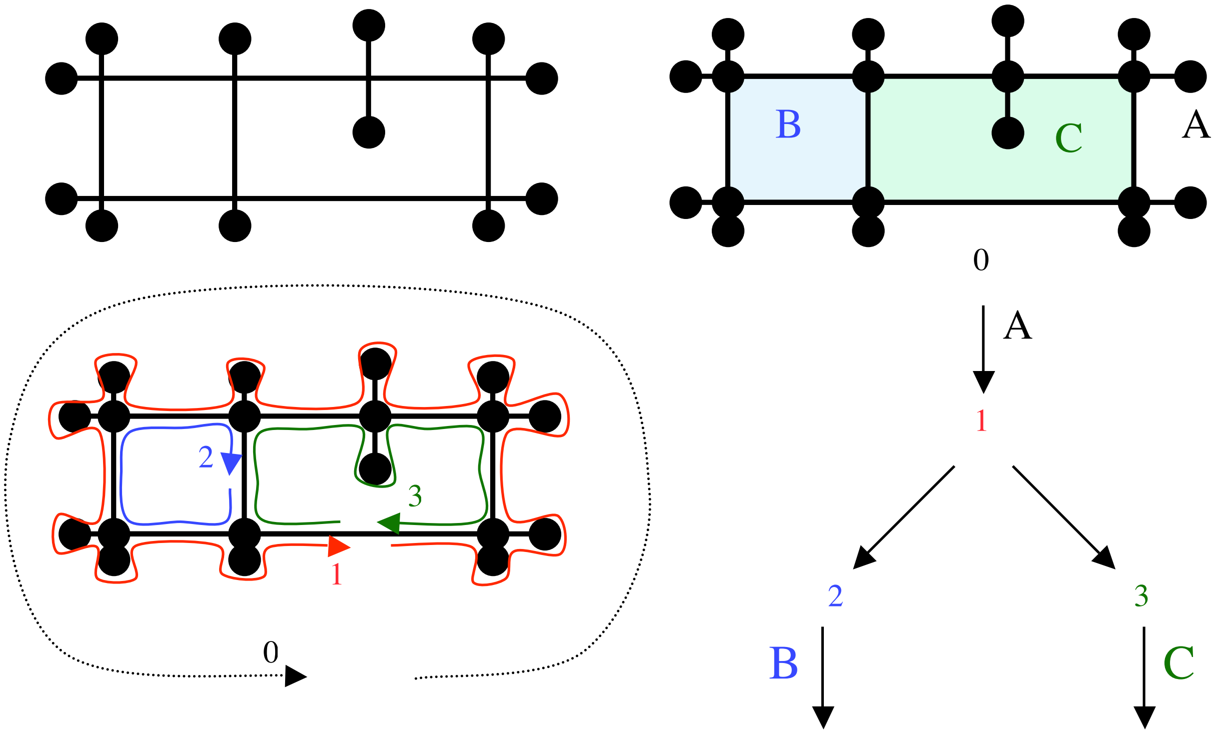


Figure 8*: An arrangement of edges (top-left) with numbered and color-coded sidewalk circuits (bottom-left). The labeled faces (top right) and the corresponding inclusion tree (bottom-right). Note that face A extends to infinity and, by definition, is bounded by the infinite sidewalk 0.*

HAUSDORFF DISTANCE

The minimum distance between sets is useful for instance to establish contact constraints, to predict collision, or to check clearances [deBe&97]. The Hausdorff distance [Atal83] measures the maximum discrepancy between two sets and is sometimes used to measure the error made during simplification [RoBo93].

Formal definitions of minimum and Hausdorff distances

Consider two sets, A and B. The **minimum distance**, M(A,B), between them is $\min_{a \in A, b \in B}(||ab||)$. The **Hausdorff distance**, H(A,B), between them is max(D(A,B),D(B,A)), where the deviation D(A,B) of A from B is $\max_{a \in A}(\min_{b \in B}(||ab||))$.

A conjecture

Although appealing, the conciseness of these definitions fails to expose the simplicity of these measures and the relation between them. For example, it makes it difficult to decide whether the conjecture below is correct.

"*Consider that A and B are triangle meshes.* $M(A,B) = \min_{a \in T(A), b \in T(B)}(M(a,b))$*, where T(A) is the set of triangles of A and T(B) the set of triangle of B. Furthermore,* $H(A,B) = \max(\max_{a \in T(A)} \min_{b \in T(B)}(H(a,b)), \max_{b \in T(B)} \min_{a \in T(A)}(H(a,b)))$*."*

The reader is invited to decide whether these two claims are correct. The exercise is useful, since the claims imply that the computation of the two distances may be reduced to the computation of miminum and Hausdorff distances between pairs of triangles, which is a very simple problem. For instance, the second claim has inspired a recently proposed, very fast, approach for computing the Hausdorff error between pairs of triangle meshes.

The romantic metaphor

To make these distance notions more intuitive and to help students to reason about them, I offer the following, more romantic, yet equivalent definitions, illustrated in Fig 9.

"Consider two territories, A and B. She was on one, I on the other. We loved each other and wanted to sleep as close as we could to each other. What separated us then was the minimum distance, M(A,B), between A and B. But things changed. Now she hates me… I still love her though, and want to get as close as possible. But she wants to sleep as far away from me as she can (presumably I snore). She chose her territory. What separates us now, is the Hausdorff distance, H(A,B), between A and B."

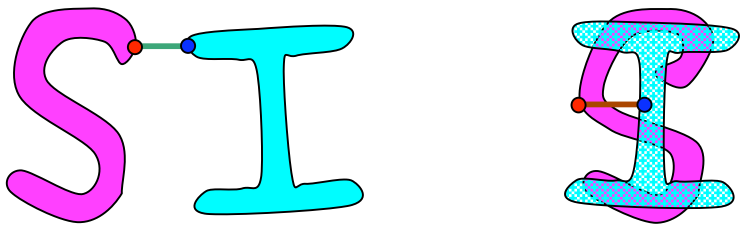


Figure 9*: When we love each other (left), the distance that separates us (green edge) is the minimum distance between the two sets. It measures the clearance between two sets. (Equivalently, it measures the maximum distance by which we could grow either set without producing an overlap.) When I love her and she hates me (right), she picks the most isolated spot (red dot). I try to get as close as possible while remaining on the other set (blue dot). The distance that separates us now (brown edge) is the Hausdorff distance. It measures the discrepancy between two sets.*

Note that when the distance is zero, whether we use the Minimum or the Hausdorff measure of distance, we sleep together. However, the implications on the relation between the two regions are drastically different: M(A,B)=0 implies that the two territories overlap (i.e. have at least one common point), while H(A,B)=0 implies A=B, otherwise, she would find a spot that I cannot reach.

Our metaphor makes it clear that the claim *"$H(A,B) = max\ (\ max_{a \in T(A)} min_{b \in T(B)}(H(a,b))\ ,\ max_{b \in T(B)} min_{a \in T(A)}(H(a,b))\ )$" is not always true*. Consider for instance the 2D territories in Fig. 10. She picks the territory that is the union of the 4 pink triangles and decides to sleep in the center of the central triangle, as far away from the blue triangles as she can. My best bet, if I want to get close to her, is then be to stand in the middle of an edge of a blue triangle. The distance that separates us is not the Hausdorff distance between our two triangles. In 3D the situation gets worse. She could settle in the middle of one of her triangles and, to get as close as possible to her, I would need to stand in the interior of one of my triangles, not on an edge or vertex. To see this, simply tilt the three blue triangles exposing their top surface to her. Thus, to compute the Hausdorff distance between two triangle meshes, it is not sufficient to consider all pairs of triangles, one from each mesh. One needs also to consider all combinations of one triangle from one set and three triangles from the other set. This conclusion indicates that without an effective pruning technique, the cost of this computation grows with the fourth power of the number of triangles in the sets, which may explain why no correct implementation of the exact Hausdorff distance between polyhedra is publicly available. Approximating solutions are based on replacing one set by a dense set of point samples [CRS98].

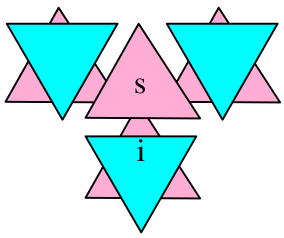


Figure 10*: She hates me and settles to sleep in the middle (s) of the central triangle. I want to be close to her and stand at the edge (i) of a blue triangle. The distance that separates us is the Hausdorff distance between the blue and the pink sets. It cannot be computed by considering only pairs of blue and pink triangles.*

DISCUSSIONS AND CONCLUSIONS

Examples of EDR techniques

The examples presented in the previous sections illustrate following EDR techniques: (1) introduce a new concept through a metaphor or analogy with a familiar situation to benefit from a previously developed intuitive understanding of the concepts and their interrelations; (2) invent an expressive notation and evocative names to facilitate memorization and to provide the students with an intuitive vocabulary for building examples, algorithms, or proofs; and (3) focus only on the essential, oversimplified concept, until is is well understood and the students are ready to address the limitations and extensions.

This approach has obvious drawbacks: (1) metaphors are imperfect and may be misleading, (2) non-standard names and notations may confuse students when they compare the course material to what is published elsewhere, and (3) the simplicity and elegance of the essentials may hide the importance of details and the difficulties involved in dealing with special cases or numeric round-off errors.

Still, an over-simplified, intuitive interpretation provides a solid backbone upon which one may attach, as minor variations or extensions, a discussion of the limitations of a particular metaphor, of the need for considering special cases, and of the possibility of generalizing or specializing the oversimplified model.

Benefits of EDR

The EDR **simplification** process may help demystify a complex subject. It helps the students grasp the main aspects quickly and enables them to put what they learn to good use immediately. For example, the ability to sit down after a lecture and program from scratch a piece of code that generates smooth curves and surfaces, and this without the need for references or even course notes, is a source of immediate gratification for a student and possibly an incentive to learn more about the subject.

The decomposition of an approach into **primitive steps** that have clear **names** ("split", "tweak") and a simple geometric interpretation ("insert a new vertex in the middle of each edge") makes it easier to develop an intuitive understanding of the behavior of the whole process, to invent variations, and to evaluate their advantages and drawbacks. For example, the idea of "Jarek's Tweak" came naturally from the realization that the B-spline subdivision tucks the old vertices "inwards" and that the 4-point subdivision pushes the new vertices "outwards". It was tempting to see what happens if we did a little of each.

The **metaphor** of the sidewalks in the computation of the plane arrangement simplifies considerably the exposition of an algorithm. It enables the student to grasp what the algorithm does at a high level, before diving into implementation details. Having a child run along a sidewalk without crossing any street is easily understood without any prior knowledge of geometric and topological principles. Clearly, the child will know where to turn. Yet, expressing such a graph traversal strategy using commonly used terminology in geometric modeling (such as edges, intersections, vertices, nodes, links, orientations, ordering) will take significantly longer and may discourage the students, or even the instructor. Once the notion of circuits was internalized, it became obvious how they can be ordered into a partial inclusion tree and how that ordering may be used to identify faces by their bounding circuits.

Finally, the romantic metaphor of the minimum and Hausdorff distances, and the terminology that it implies, make for a more **entertaining** lecture and ease **communication**. It also helps ensure that a student will remember these definitions and their differences.

What needs to be done about EDR

EDR needs to be recognized as a true Research activity. Its impact is as important as the technical research itself, since it enables students and researchers to better understand and use prior research results. To do so, we must learn how to evaluate EDR contributions and carve a space for them in peer-reviewed technical journals. Finally, we need to draw from the vast body of research in education.

Acknowledgement

Thanks go to Webb Roberts for producing the first implementation of the Split&Tweak subdivision schemes (Fig. 4) and to Jessie Shieh for providing the web-based applet which was used to generate Figs. 5 and 6.

References

[Atal83] M. J. Atallah. A linear time algorithm for the Hausdoff distance between convex polygons. Inform. Process. Lett., 17:207–209, 1983.

[BaGo88] P. Barry and R. Goldman, A Recursive Evaluation Algorithm for a Class of Catmull-Rom Splines, Computer Graphics, Vol. 22, No. 4, p. 199-204, August 1988.

[CRS98] P. Cignoni, C. Rocchini and R. Scopigno, "Metro: measuring error on simplified surfaces", Proc. Eurographics '98, vol. 17(2), pp 167-174, June 1998.

[deBe&97] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf, Computational Geometry - Algorithms and Applications. Springer-Verlag. 1997

[Fari93] G. Farin, (1993), *Curves and Surfaces for CAGD*, Academic Press, Boston. 1993.

[RoBo93] J. Rossignac and P. Borrel, "Multi-resolution 3D approximations for rendering complex scenes", Geometric Modeling in Computer Graphics, Springer Verlag, Berlin, pp. 445-465, 1993.