Geometric data structures for multihierarchical XML tagging of manuscripts

Jerzy W. Jaromczyk [a,1],Neil Moore [a,1],

[a]Deparment of Computer Science, University of Kentucky, Lexington, KY, USA

Abstract

This paper shows an application of computational geometry methods to the preparation of image-based digital library editions. We present a formalism for describing non-hierarachical markup of manuscripts in terms of one- and two-dimensional geometry. This formalism allows us to use geometric data structures and algorithms to process marked-up documents. With this approach we can overcome many well-known and inherently difficult problems with non-hierarchical markup. We present algorithms based on segment trees and range-query structures for performing a number of queries on markup structure. This application of computational geometry data structures to a new domain also provides insights into novel types of geometric operations and queries. Some of these techniques are currently being used by researchers in the Research Computing for Humanities project, which aims to produce electronic editions of selected manuscripts from the British Library.

1. Introduction

From a computational geometry point of view, an old manuscript, a *folio* (a sheet of writing material) and the script (a text on the manuscript page) have interesting features in one, two and three dimensions and all of them are important. Spatial deformations of the folio can be studied for restoration purposes in three dimensions. Illuminations, damages, restorations, and paleographic features of individual letters can be studied as two dimensional objects. The script, a sequence of lines of text, can be viewed as one dimensional as it follows visible or invisible rulings that guide the layout of the text [B94]. In fact, connecting the image view of a manuscript with its transcript is typically the first task. In this paper we will discuss this linear aspect of folia and we will discuss geometric structures that support its tagging.

Extensive tagging or markup is an often excruciating task in the preparation of electronic editions of manuscripts. The tagging process results in a structured description of the document's contents, its features and attributes in a form that can be used to view and effectively query the edition. The XML (eXtensible Markup Language) is a prevailing format used for describing literary and artistic work for Digital Libraries. XML files describe well-hierarchical structures (e.g., book, volume, section, page, line, word and character) of the document and for that reason they can be viewed as rooted trees. Although it seems natural that most documents adhere to such hierarchies, it is often not true in practice. Even worse, this happens in the context of cultural heritage that urgently requires preservation: old and severely damaged manuscripts. An image of a damaged folio from Alfred the Great's Old English translation of Boethius's *Consolation of Philosophy* is demonstrated in Figure 1.

Words can span more than one line, damages or restorations can overlap words and parts of them. A rather simple case is illustrated in Figure 2 where in the convoluted tagging one word spans two lines; as such, it is not well-formed XML.

It has been long recognized that, in spite of its popularity, XML suffers from an inability to encode elements that are not in hierarchical relationships [RMD93]. There are numerous approaches to address this problem called the *concurrent hierarchies* problem; see, for example, [B95]. Mostly these approaches are concerned with tagging tran-

Email addresses: jurek@cs.uky.edu (Jerzy W. Jaromczyk), neil@cs.uky.edu (Neil Moore).

[1] This research was supported in part by NSF ITR grant 0219924

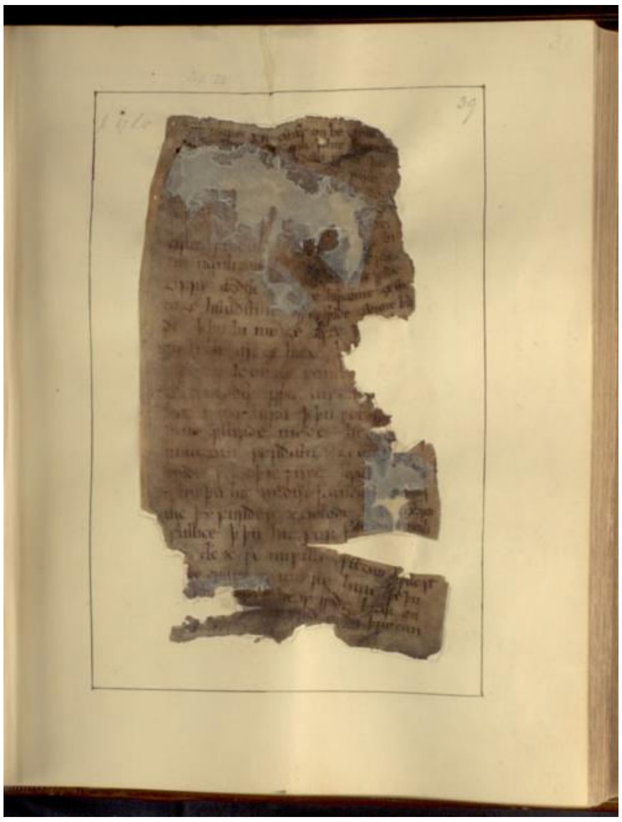
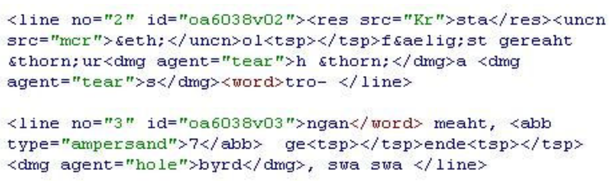


Fig. 1. An image of a damaged manuscript page



```
<line no="2" id="oa6038v02"><res src="Kr">sta</res><uncn
src="mcr">&eth;</uncn>ol<tsp></tsp>f&aelig;st gereaht
&thorn;ur<dmg agent="tear">h &thorn;</dmg>a <dmg
agent="tear">s</dmg><word>tro- </line>

<line no="3" id="oa6038v03">ngan</word> meaht, <abb
type="ampersand">7</abb>  ge<tsp></tsp>ende<tsp></tsp>
<dmg agent="hole">byrd</dmg>, swa swa </line>
```

Fig. 2. Markup that is not hierarchical

scripts (text-based documents). In our task a manuscript or an image of it is the primary source for preparing an electronic edition and we need to handle the concurrent hierarchies in geometric context of the image. This image-based approach is the most distinguishing aspect of our work.

A geometric view of the problem allows us to engage many data structures, in particular multi-dimensional ones. In this paper, we will focus on two of them: segment trees and a grid based data structure developed by Overmars for range queries (we will use it for line segments rather than points, though). We will analyze how these structures support a variety of queries that are essential in the context of editing and studying manuscripts.

There are several contributions of this paper. The first is in applying computational geometry to a new area. We will present a formalism for concurrent hierarchies that allows us to connect geometric structures with XML documents. Specifically, we will discuss a number of algorithms for querying the structure of a document, together with their asymptotic complexities.

2. Definitions

In this section we describe a formalism for representing multihierarchical document markup. This formalism allows us to connect the structure of markup with a geometric representation; this allows us to use geometric data structures and algorithms to process tagged documents.

2.1. *Markup elements*

We represent a markup element as a tuple (N, A, α, ω) where N is the element name, A (a map from strings to strings) the attributes, and α and ω are integers with $1 \leq \alpha \leq \omega$. The interval of a markup element e, written $\mathrm{I}(e)$, is the closed interval $[\alpha(e), \omega(e)] \subset \mathbb{N}$.

Definition **2.1** *Let e_1 and e_2 be two markup elements. We define the relations:*

- *$e_1 \prec e_2$ if $\omega(e_1) < \alpha(e_2)$*
- *$e_1 \sqsubset e_2$ if $\alpha(e_1) \geq \alpha(e_2)$ and $\omega(e_1) \leq \omega(e_2)$ and $\mathrm{I}(e_1) \neq \mathrm{I}(e_2)$. In this case we say that e_1 is a descendant of e_2, or equivalently that e_2 is an ancestor of e_1.*

We state without proof the following theorems:

Theorem **2.1** *The relations $\prec$ and $\sqsubset$ each form a strict partial order.*

Theorem **2.2** *Let e_1 and e_2 be two markup elements. Exactly one of the following holds: $e_1 \prec e_2$; $e_2 \prec e_1$; $e_1 \sqsubset e_2$; $e_2 \sqsubset e_1$; e_1 overlaps e_2; or $\mathrm{I}(e_1) = \mathrm{I}(e_2)$.*

3. Hierarchies

Definition **3.1** *Let E be a finite set of markup elements. E is hierarchical if:*

- *There is an element $r \in E$, called the root, such that, for each element $e \in E$, $e \sqsubset r$ or $e = r$*
- *No two elements of E overlap, and no distinct elements of E share the same interval.*

Lemma **3.1** *Let E be a hierarchical set of markup elements, and $e \in E$. Then either e is the root and has no parents in E, or e is not the root and has exactly one parent in E.*

THEOREM **3.1** *Let E be a hierarchical set of markup elements. Let G be a graph on E such that the edge (e_1, e_2) is in G if e_1 is the parent of e_2. G is a tree.*

This justifies our use of the term "hierarchy". Because children of the same parent cannot be descendants of one another, by Theorem 2.2 they can be ordered by $\prec$. The tree is therefore an ordered tree.

4. Data structures and queries

In this section we describe two geometric data structures and apply them to a number of common queries on documents.

A segment tree [BW80, PS85] is a dynamic data structure to represent a set of segments. For insertions and deletions it is assumed that the endpoints belong to the set of n points known in advance.

The underlying structure is a balanced binary tree with leaves representing atomic segments. Each node corresponds to the union of the atomic segments rooted in this nodes. Intervals that belong to the collection represented in the segment tree are associated with nodes of the tree and satisfy the following property: a node v stores s if the union of its atomic segments is contained in s but the union of the atomic segments associated with the parent of v do not. Thanks to this property each segment is represented in at most $\mathcal{O}(\log n)$ nodes.

Insertions and deletions can be performed in $\mathcal{O}(\log n)$ time. Also, in the same time one can count the number of segments in the collection that includes a given query point.

While segment trees are well-suited for some types of queries, there are other queries which segment trees do not perform as efficiently. We use a range- query structure to support these queries.

We treat a segment $S = [\alpha, \omega]$ in $U = [1, M]$ as a point $p(S) = (\alpha, \omega)$ in U^2. We call this representation of (a collection of) segments a *segment grid.* Many properties of S then correspond to range properties of $p(S)$ in the segment grid. For example, $S \subset T$ if and only if $p(S)$ lies to the lower right of $p(T)$.

We shall make use of general range queries of the form: find all points lying in the (closed) rectangle bounded by (a, b) and (c, d). There exist a number of data structures supporting such queries in a two-dimensional grid. A number of these structures are described in [O88]; two are of particular interest for our purposes.

THEOREM **4.1** *(Overmars) We can represent n points in U^2 (and thus n segments in U) using $\mathcal{O}(n \log n)$ space in such a way that range queries take $\mathcal{O}(k + \log\log |U|)$ time, where k is the number of results returned by the query.*

This data structure makes use of perfect hashing, and is therefore slow to build. There is an alternative data structure with slightly worse query time, but significantly better creation time:

THEOREM **4.2** *(Overmars) We can represent n points in U^2 using $\mathcal{O}(n \log n)$ space in such a way that range queries take $\mathcal{O}\left(k + \sqrt{\log |U|}\right)$ time, where k is the number of returned results. This data structure can be built in $\mathcal{O}(n \log n)$ time.*

Neither of the range-query data structures permits efficient insertion or deletion. For more information on these structures, see [O88].

4.1. *Descendant queries*

A common query on documents is to find all elements of a certain type that are descendants of a given element e. For example, given a `<page>` element, one may wish to find all `<damage>` elements contained within that element, either directly (as children) or indirectly. We can perform this operation by finding all descendants of e and reporting only those of the requested type.

Recall from Definition 2.1 that the descendants of e are those elements which begin no earlier than e, end no later than e, and do not both begin and end at the same point as e. In terms of the segment grid, $p(\mathrm{I}(x))$ is a descendant of e if $\mathrm{I}(x) \neq \mathrm{I}(e)$ and $p(\mathrm{I}(x))$ lies in the rectangle bounded by the points $(\alpha(e), \alpha(e))$ and $(\omega(e), \omega(e))$. Using the data structure from Theorem 4.2, we can find all such points in $\mathcal{O}(k + \sqrt{\log M})$ time, where M is the document's maximum offset and k is the number of results.

4.2. *Overlap queries*

Another useful query is: given an element e, find all elements which overlap e. If e_1 overlaps e_2, e_1 contains at least one of the endpoints of e_2. Conversely, if e_1 contains at least one endpoint of e_2, either e_1 overlaps e_2, $e_1 = e_2$, $e_1 \sqsubset e_2$, or $e_2 \sqsubset e_1$. This suggests the following:

THEOREM **4.3** *Let D be a document containing the element e. Let k be the number of elements overlapping e, d the number of descendants of e, a the number of ancestors of e, and M the maximum off-*

set of D. We can find all elements overlapping e in $\mathcal{O}(k + d + a + \log M)$ time.

We first find the sets $B(e)$ and $E(e)$ of all elements whose intervals contain $\alpha(e)$ and $\omega(e)$, respectively, using stabbing queries in a segment tree. $B(e)$ contains at most $k + d + a + 1$ elements, and likewise for $E(e)$. The stabbing queries can therefore be performed in time $\mathcal{O}(k+d+a+\log M)$. We then iterate through the results, reporting those which actually overlap e. Testing whether a given segments overlaps e requires constant time, so this step does not increase the complexity.■

We can improve on this bound somewhat by making use of range queries. If e_1 overlaps e_2, e_1 contains *exactly* one endpoint of e_2. In the segment grid, segments containing $\alpha(e)$ but not $\omega(e)$ lie in the rectangle R_α bounded by the points $(1, \alpha(e))$ and $(\alpha(e), \omega(e))$. Likewise, segments containing $\omega(e)$ but not $\alpha(e)$ lie in the rectangle R_ω bounded by $(\alpha(e), \omega(e))$ and $(\omega(e), M)$, where M is the maximum offset of the document. While these rectangles contain all the elements which overlap e, they do not contain *only* such elements. As with the stabbing queries described above, the range queries also return e, and may return some ancestors and descendants of e, so we must remove these from the result set. It is clear, however, that the rectangles do not contain any element x such that $x \prec e$ or $e \prec x$.

THEOREM **4.4** *Let D be a document containing the element e. Let k be the number of elements overlapping e, d the number of descendants of e, a the number of ancestors of e, and M the maximum offset of D. We can find all elements overlapping e in $\mathcal{O}(k + d + a + \sqrt{\log M})$ time.*

Each range query returns at most $k + d + a + 1$ elements, and can therefore be performed in time $\mathcal{O}(k+d+a+\sqrt{\log M})$. As with the stabbing query, we then iterate through the results of the range query, reporting those elements which overlap e; again, this does not affect the overall complexity of the overlap query.■

The running time of overlap queries can be improved still further if we impose additional restrictions on the endpoints of elements:

THEOREM **4.5** *Let D be a document such that no two elements share an endpoint in common, and let e be an element in D. If k is the number of elements overlapping e and M is the maximum offset of D, we can find all elements overlapping e in $\mathcal{O}(k + \sqrt{\log M})$ time.*

5. Conclusion

We have presented a formalism that connects the realm of computational geometry to algorithmic problems that we have faced in the marking up of image-based electronic editions. As examples of geometric structures we have discussed segment trees and range query structures and have applied them to the well-recognized and inherently difficult problem of non-hierarchical markup. Two-dimensional aspects of manuscript marking, such as marginalia, paleographic features and damages will involve additional geometric structures.

Acknowledgement

The authors acknowledge a partial support from the NSF ARCHway: Architecture for Research in Computing for Humanities through Research, Teaching and Learning project. We are thankful to Kevin Kiernan, the Boethius Project and the British Library Board for allowing us to use the image. We also thank our colleagues from the Research Computing for Humanities Lab at the University of Kentucky for many useful discussions.

Bibliography

[B95] Barnard, D., et al. Hierarchical Encoding of Text: Technical problem and SGML Solutions, *Computers and the Humanities* 29/3 (1995) 211-231.

[BW80] Bentley, J. L., and D. Wood. An optimal worst-case algorithm for reporting intersections of rectangles, *IEEE Trans. on Computers* C-29 (1980) 571-577.

[B94] Brown, M. P. *Understanding Illuminated Manuscripts: a Guide to Technical Terms.* London: The British Library, 1994.

[KJDP03] Kiernan, K., J. W. Jaromczyk, A. Dekhtyar, D. C. Porter, et al. The ARCHway Project: Architecture for Research in Computing for Humanities through Research, Teaching, and Learning. To be published in *Literary and Linguistic Computing*, 2003.

[O87] Overmars, Mark H. Efficient data structures for range searching on a grid, *J. Algorithms* 9 (1988) 254-275.

[PS85] Preparata, F. P. and M. I. Shamos. *Computational Geometry: an Introduction.* New York: Springer-Verlag, 1985.

[RMD93] Renear, A., E. Mylonas, and D. Durand. Refining our Notion of What Text Really Is: The Problem of Overlapping Hierarchies. In *Research in Humanities Computing*, eds. N. Ide and S. Hockey. Oxford: Oxford Univeristy Press, 1993.

[S99] Samet, H.. Multidimensional Data Structures. In *Algorithms and Theory of Computation Handbook*, ed. M. J. Atallah. Boca Raton: CRC Press, 1999.

[XML-W3C] Bray, T., J. Paoli, C. M. Sperberg-McQueen, and E. Maler, eds. Extensible Markup Language (XML) 1.0, W3C Recommendation. `http://www.w3.org/TR/2000/REC-xml-20001006`. W3 Consortium, 2000.