Abstract: Different embodiments of the invention use different techniques for analyzing an unstructured document to define a structured document. The unstructured document includes numerous primitive elements, but does not include structural elements that specify the structural relationship between the primitive elements and/or structural attributes of the document based on these primitive elements. To define the structured document, the primitive elements of the unstructured document are used to identify various geometric attributes of the unstructured document. The identified geometric attributes and other attributes of the primitive elements are used to define structural elements, such as associated primitive elements (e.g., words, paragraphs, joined graphs, etc.), tables, guides, gutters, etc, as well as to define the flow of reading through the primitive and structural elements. Various methods to enhance the efficiency of the geometric analysis and document reconstruction processes, (e.g., hierarchical profiling, efficient cluster analysis techniques, efficient data structures) are provided.
Published:
— without international search report and to be republished upon receipt of that report (Rule 48.2(g))
METHODS AND SYSTEM FOR DOCUMENT RECONSTRUCTION

FIELD OF THE INVENTION

[0001] The invention is directed towards document reconstruction. Specifically, the invention is directed towards methods for defining a structured document from an unstructured document, for improving the efficiency of such processes, and for improving display of and interaction with structured documents.

BACKGROUND OF THE INVENTION

[0002] Documents are often defined as nothing more than a collection of primitive elements that are drawn on a page at defined locations. For example, a PDF (portable document format) file might have no definition of structure and instead is nothing more than instructions to draw glyphs, shapes, and bitmaps at various locations.

[0003] A user can view such a document on a standard monitor and deduce the structure. However, because such a file is only a collection of primitive elements, a document viewing application has no knowledge of the intended structure of the document. For example, a table is displayed as a series of lines and/or rectangles with text between the lines, which the human viewer recognizes as a table. However, the application displaying the document has no indication that the text groupings have relationships to each other based on the rows and columns because the document does not include such information. Similarly, the application has no indication of the flow of text through a page (e.g., the flow from one column to the next, or the flow around an embedded image), or various other important qualities that can be determined instantly by a human user.

[0004] This lack of knowledge about document structure will not always be a problem when a user is simply viewing the document on a standard monitor. However, it would often be of value to a reader to be able to access the file and edit it as though it were a document produced by a word processor, image-editing application, etc., that has structure and relationships between elements.

[0005] Therefore, there is a need for methods that can reconstruct an unstructured document. Similarly, there is a need for methods that take advantage of such reconstructed
document structure to idealize the display of the document (e.g., for small-screen devices where it is not realistic to display the entire document on the screen at once), or to enable intelligent selection of elements of the document.

[0006] In the modern world, more and more computing applications are moving to handheld devices (e.g., cell phones, media players, etc.). Accordingly, document reconstruction techniques must be viable on such devices, which generally have less computing power than a standard personal computer. However, document reconstruction often uses fairly computation and memory intensive procedures, such as cluster analysis, and the use of large chunks of memory. Therefore, there is further a need for techniques that allow for greater efficiency in document reconstruction generally, and cluster analysis specifically.
SUMMARY OF THE INVENTION

[0007] Different embodiments of the invention use different techniques for analyzing an unstructured document to define a structured document. In some embodiments, the unstructured document includes numerous primitive elements, but does not include structural elements that specify the structural relationship between the primitive elements and/or structural attributes of the document based on these primitive elements. Accordingly, to define the structured document, some embodiments use the primitive elements of the unstructured document to identify various geometric attributes of the unstructured document, and then use the identified geometric attributes and other attributes of the primitive elements to define structural elements, such as associated primitive elements (e.g., words, paragraphs, joined graphs, etc.), tables, guides, gutters, etc., as well as to define the flow of reading through the primitive and structural elements.

[0008] As mentioned, some embodiments use primitive elements to identify various geometric attributes. For instance, some embodiments provide a method that identifies boundaries between sets of primitive elements and regions bounded by the boundaries. The method uses the identified regions to define structural elements for the document, and defines a structured document based on the primitive elements and the structural elements. In some embodiments, defining structural elements includes analyzing each region separately to create associations between sets of primitive elements in the particular region. In some embodiments, defining the structured document includes identifying hierarchical relationships between the identified regions.

[0009] Some embodiments provide a method that analyzes an unstructured document that includes numerous words, where each word is an associated set of glyphs and each glyph has location coordinates. The method identifies clusters of location values, where each location value is associated with one word, is a basis for word alignment, and is derived from the location coordinates of the glyphs of that word. Based on the identified clusters of location values, the method defines a set of boundary elements for the words that identify a set of alignment guides for the words. The method defines a structured document based on the glyphs and the defined boundary elements. Some embodiments also define at least one region of white space between a pair of boundary elements and further define the structured document based on the region of
white space. Some embodiments identify the clusters of location values by using density clustering.

Some embodiments use the identified geometric attributes and other attributes of the primitive elements to define structural elements as well as to define the flow of reading through the primitive and structural elements. For instance, some embodiments provide a method that analyzes an unstructured document that includes numerous glyphs, each of which has a position in the unstructured document. Based on the positions of glyphs, the method creates associations between different sets of glyphs in order to identify different sets of glyphs as different words. The method creates associations between different sets of words in order to identify different sets of words as different paragraphs. The method defines associations between paragraphs that are not contiguous in order to define a reading order through the paragraphs. In order to create associations between different sets of words in order to identify different sets of words as different paragraphs, some embodiments create associations between different sets of words as different text lines, and create associations between different sets of text lines as different paragraphs.

Some embodiments provide a method that identifies boundaries between sets of glyphs and identifies that several of the boundaries form a table. The method defines a tabular structural element based on the table that includes several cells arranged in several rows and columns, where each cell includes an associated set of glyphs. Some embodiments identify that the boundaries form a table by identifying a set of boundaries that form a larger rectangular shape and several rectangular shapes contained within the larger rectangular shape. In some embodiments, at least some of the identified boundaries are inferred based on positions of the associated sets of glyphs that form the cells.

Some embodiments provide a method for analyzing an unstructured document that includes numerous primitive graphic elements, each of which is defined as a single object. The document has a drawing order that indicates the order in which the primitive graphic elements are drawn. The method identifies positional relationships between successive primitive graphic elements in the drawing order. Based on the positional relationships, the method defines a single structural graphic element from several primitive graphic elements. Some embodiments identify a positional relationship between a first and second primitive graphic element that arc
subsequent in the drawing order by calculating a size of a structural graphic element that includes the first and second primitive graphic elements.

[0013] Some embodiments provide methods to make geometric analysis and document reconstruction more effective. For instance, some embodiments provide a method that provides a default set of document reconstruction operations for defining a structured document that comprises a plurality of primitive elements. The method provides a hierarchical set of profiles, each profile including (i) a set of document reconstruction results and (ii) results for modifying the document reconstruction operations when intermediate document reconstruction results match the potential document reconstruction results for the profile. Instructions from a profile at a lower level in the hierarchy override instructions from a profile at a higher level. In some embodiments, the instructions for a particular profile include a subset of profiles at a lower level in the hierarchical set of profiles that should be tested when the intermediate document reconstruction results match the potential document reconstruction results for the profile.

[0014] Once a structured document is defined, some embodiments provide various techniques for idealizing user interaction with the structured document. For instance, some embodiments provide a method for displaying a structured document that includes a hierarchy of structural elements constructed by analyzing an unstructured document. The method displays the structured document on the device (e.g., a small-screen device). The method receives a position of interest in the document, and identifies a structural element within the hierarchy as a region of interest based on the position of interest. The method modifies the display of the document to highlight the identified region of interest. Some embodiments identify the structural element by identifying a structural element at the lowest level of the hierarchy that includes the position of interest, and identifying structural elements at higher levels of hierarchy that include the structural element identified at the lowest level until a structural element qualifying as a region of interest is reached. Some embodiments also receive an input to move from the region of interest and modify the display of the document to highlight a structurally related region of interest.

[0015] Some embodiments provide a method for defining a selection of text in an unstructured document that includes numerous glyphs. The method identifies associated sets of glyphs and a reading order that specifies a flow of reading through the glyphs. The method
displays the document and receives a start point and end point for a selection of text within the displayed document. The method defines the selection of text from the start point to the end point by using the identified sets of glyphs and intended flow of reading. In some embodiments, the associated sets of glyphs are paragraphs and the reading order specifics a flow of reading from a first paragraph to a second paragraph that are not contiguous.

[0016] Some embodiments provide methods that enhance the efficiency of the geometric analysis and document reconstruction processes. Some embodiments use cluster analysis for geometric analysis and/or document reconstruction, which can be a computing-intensive process. Accordingly, some embodiments provide a method that defines structure for an unstructured document that includes numerous primitive elements that are defined in terms of their position in the document. The method identifies a pairwise grouping of nearest primitive elements and sorts the pairwise primitive elements based on an order from the closest to the furthest pairs. The method stores a single value that identifies which of the pairwise primitive elements are sufficiently far apart to form a partition. The method uses the stored value to identify and analyze the partitions in order to define structural elements for the document.

[0017] Some embodiments also provide methods for making use of efficient data structures. For instance, some embodiments provide several different processes for analyzing and manipulating an unstructured document that includes numerous primitive elements. Some embodiments also provide a storage for data associated with the primitive elements. At least some of the data is stored in a separate memory space from the processes and is shared by at least two different processes. The processes access the data by use of references to the data. The data is not replicated by the processes.
BRJEF DESCRIPTION OF THE DRAWINGS

[0018] The novel features of the invention are set forth in the appended claims. However, for purposes of explanation, several embodiments of the invention are set forth in the following figures.

[0019] Figure 1 illustrates the overall reconstruction flow of some embodiments.

[0020] Figure 2 illustrates a page of a document and various results from geometric analysis and document reconstruction of some embodiments being performed on the page.

[0021] Figure 3 conceptually illustrates a process of some embodiments for identifying zones of a page of a document and generating a zone tree for the page.

[0022] Figure 4 illustrates a page and a sequence of identifying zones of the page and generating a zone tree for the page in some embodiments.

[0023] Figure 5 illustrates a page of a document that includes several zones.

[0024] Figure 6 illustrates a page that includes zone border graphics and multiple zones, including rotation groups.

[0025] Figure 7 illustrates a zone tree of some embodiments for the page from Figure 5.

[0026] Figure 8 conceptually illustrates a process of some embodiments for defining rotation groups on a page.

[0027] Figure 9 conceptually illustrates a process of some embodiments for identifying zone borders and intersections.

[0028] Figure 10 illustrates a page that includes various graphics and text.

[0029] Figure 11 illustrates the zone border intervals and intersections for the page of Figure 10.

[0030] Figure 12 conceptually illustrates a process of some embodiments for identifying zones.

[0031] Figures 13 and 14 illustrate the application of the process of Figure 12 to identify the zones of the page of Figure 10.
Figure 15 conceptually illustrates a process of some embodiments for generating a zone tree.

Figure 16 illustrates the zones from the page of Figure 10 sorted by size and placed into a node graph.

Figure 17 conceptually illustrates the software architecture of a zone analysis application of some embodiments.

Figure 18 illustrates an overall process of some embodiments for identifying guides and gutters in a document.

Figure 19 illustrates a page having two columns of text, and the guides and gutters identified for the page.

Figure 20 conceptually illustrates a process of some embodiments for performing density clustering.

Figure 21 conceptually illustrates a process of some embodiments for determining left-alignment guides.

Figures 22-24 illustrate the identification a left-alignment guide on a page.

Figure 25 conceptually illustrates a process of some embodiments for determining right-alignment guides.

Figure 26 conceptually illustrates a process of some embodiments for determining gutters for a region of a document.

Figures 27-29 illustrate the identification of a gutter on a page.

Figure 30 conceptually illustrates the software architecture of a guide and gutter analysis application of some embodiments.

Figure 31 conceptually illustrates a process of some embodiments for determining the layout and flow of a document.

Figure 32 illustrates a sequence of some embodiments of the determination of layout and flow information for a page of a document.
[0046] Figure 33 conceptually illustrates a process of some embodiments for identifying and merging lines of text.

[0047] Figure 34 illustrates a page with six groups of overlapping text lines.

[0048] Figure 35 illustrates the merging of the groups of text lines from Figure 34.

[0049] Figure 36 conceptually illustrates a process of some embodiments for performing difference clustering.

[0050] Figure 37 illustrates an example of difference clustering.

[0051] Figure 38 conceptually illustrates a process of some embodiments for splitting lines of text.

[0052] Figure 39 illustrates a sequence showing the identification of where to split lines of text on a page.

[0053] Figure 40 conceptually illustrates a process of some embodiments for grouping text lines into paragraphs.

[0054] Figure 41 illustrates the identification of paragraphs on a page.

[0055] Figure 42 conceptually illustrates a process of some embodiments for identifying columns and layouts in a portion of a document.

[0056] Figures 43 and 44 illustrate paragraphs on two different pages.

[0057] Figures 45 and 46 illustrate the generation of flow graphs for the pages of Figures 43 and 44.

[0058] Figure 47 conceptually illustrates the software architecture of a layout and flow analysis application of some embodiments.

[0059] Figure 48 conceptually illustrates a process of some embodiments for identifying tables in a portion of a document.

[0060] Figure 49 illustrates an example of a table with complete borders.

[0061] Figure 50 illustrates an example of a table with connected but incomplete borders.
Figure 51 illustrates an example of a table with no borders.

Figure 52 illustrates an example of a table with disconnected borders.

Figure 53 conceptually illustrates a process of some embodiments for identifying tables that have a complete set of borders and defining the cells, rows, and columns of the identified tables.

Figure 54 illustrates a page that includes a table with complete borders.

Figure 55 conceptually illustrates a process of some embodiments for identifying and reconstructing tables that have a connected set of borders.

Figure 56 illustrates a sequence showing the identification and reconstruction of a table with a connected set of borders.

Figure 57 conceptually illustrates a process of some embodiments for identifying and reconstructing tables with no borders or disconnected borders.

Figure 58-60 illustrate the application of the process of Figure 57 to a page to identify a table having disconnected borders.

Figure 61 conceptually illustrates the software architecture of a table identification application of some embodiments.

Figure 62 conceptually illustrates a process of some embodiments for joining individual graphs into joined graphs.

Figure 63 illustrates the joining of graphs on a page.

Figure 64 conceptually illustrates a process of some embodiments for performing bounds clustering to identify graphs that should be joined and joining those graphs.

Figure 65 illustrates two pages, each having two graphic objects for which the spread is calculated.

Figure 66 illustrates a process of some embodiments for processing a cluster into subsequences.
[0076] Figure 67 conceptually illustrates a graph joining application of some embodiments for identifying graphs that should be joined and associating the graphs as one graphic.

[0077] Figure 68 conceptually illustrates a process of some embodiments that uses profiles for tailoring content processing to content type.

[0078] Figure 69 illustrates a document fitting the profile of a #10 envelope.

[0079] Figure 70 illustrates a hierarchical set of profiles of some embodiments.

[0080] Figure 71 conceptually illustrates a process of some embodiments for processing content according to hierarchical profiles.

[0081] Figure 72 conceptually illustrates a process of some embodiments for performing document reconstruction using hierarchical profiling.

[0082] Figure 73 illustrates how some embodiments identify the lowest level active profile that implements a particular method.

[0083] Figures 74-77 illustrate an example page profile matching process of some embodiments for four different pages.

[0084] Figure 78 conceptually illustrates a process for displaying a document based on an identification of a position of interest in some embodiments.

[0085] Figure 79 illustrates a sequence on a small-screen device in which a position of interest is selected and the display is modified in accordance with some embodiments of the invention.

[0086] Figure 80 illustrates a first detailed process of some embodiments for identifying a region of interest and applying transformations to draw the region of interest on a display device.

[0087] Figure 81 illustrates a second detailed process of some embodiments for identifying a region of interest and applying transformations to draw the region of interest on a display device.
[0088] Figure 82 conceptually illustrates a process of some embodiments for displaying a semantically reconstructed document.

[0089] Figure 83 illustrates an application of the process of Figure 82 of some embodiments.

[0090] Figure 84 conceptually illustrates a process of some embodiments for navigating between regions in a semantically reconstructed document.

[0091] Figure 85 conceptually illustrates a process of some embodiments for modifying a layout of a semantically reconstructed document for a small-screen device.

[0092] Figure 86 illustrates a page of a document as it would be displayed on a large screen and the display of a portion of the page on a small-screen device according to some embodiments of the invention.

[0093] Figure 87 conceptually illustrates a process of some embodiments for determining a selection of text in a semantically reconstructed document.

[0094] Figure 88 illustrates a sequence of the selection of text on a page of a semantically reconstructed document according to some embodiments.

[0095] Figure 89 illustrates a first detailed process of some embodiments for defining a selection of text in a semantically reconstructed document.

[0096] Figure 90 illustrates a first start point and end point for a selection on a page of a semantically reconstructed document.

[0097] Figure 91 illustrates a selection defined by the start and end points of Figure 90 according to the process of Figure 89.

[0098] Figure 92 illustrates a second start point and end point for a selection on a page of a semantically reconstructed document.

[0099] Figure 93 illustrates a selection defined by the start and end points of Figure 92 according to the process of Figure 89.

[0100] Figure 94 illustrates a second detailed process of some embodiments for defining a selection of text in a semantically reconstructed document.
Figure 95 illustrates a selection defined by the start and end points of Figure 90 according to the process of Figure 94.

Figure 96 illustrates a selection defined by the start and end points of Figure 92 according to the process of Figure 94.

Figure 97 conceptually illustrates a process of some embodiments for semantically reconstructing a document on a limited-resource device using cluster analysis.

Figure 98 illustrates a sequence of some embodiments by which a document is semantically reconstructed.

Figure 99 conceptually illustrates a process of some embodiments for partitioning a data set by using indirectly sorted arrays.

Figure 100 illustrates the partitioning of a data set with nine data items.

Figure 101 conceptually illustrates a process of some embodiments for performing cluster analysis at multiple distance scales concurrently.

Figure 102 conceptually illustrates the software architecture of a cluster analysis application of some embodiments for performing cluster analysis.

Figure 103 conceptually illustrates a process of some embodiments for reconstructing a document efficiently.

Figure 104 illustrates a sequence by which a document is parsed and analyzed according to the process of Figure 103.

Figure 105 illustrates the manner in which data is stored according to some embodiments of the invention.

Figure 106 conceptually illustrates an API that performs document reconstruction processes while using efficient memory management techniques.

Figure 107 conceptually illustrates the software architecture of an application of some embodiments for reconstructing, displaying, and interacting with a document.
[0014] Figure 108 conceptually illustrates a process of some embodiments for manufacturing a computer readable medium that stores a computer program such as the application described in Figure 107.

[0015] Figure 109 conceptually illustrates a computer system with which some embodiments of the invention are implemented.
DETAILED DESCRIPTION OF THE INVENTION

[0016] In the following description, numerous details are set forth for purpose of explanation. However, one of ordinary skill in the art will realize that the invention may be practiced without the use of these specific details. For instance, in some cases, the techniques described below are described as taking place in a specific order. However, in some embodiments, the techniques are performed in an order different from that described. Furthermore, while the techniques are described for languages that are read left-to-right (e.g., English), one of ordinary skill will recognize that the techniques are easily adapted for right-to-left languages.

1. OVERVIEW

[0017] Some embodiments of the invention provide novel methods for defining a structured document from an unstructured document. In some embodiments, an unstructured document is a document defined to include only primitive elements such as shapes (e.g., vector graphics), images (e.g., bitmaps), and glyphs. In some embodiments, a glyph is a visual representation of a text character (e.g., a letter, a number, a punctuation mark, or other inline character), collection of characters, or portion of a character. In some embodiments, a glyph may be a pre-specified collection of scalable vector graphics including path definitions for the outline of the glyph. In some embodiments, a glyph may be a pre-specified raster image or collection of raster images optimized for various sizes. As an example, the character "i" could be represented by a single glyph that is a path with two sub-paths, one for the outline of the dot and one for the outline of the lower portion. As another example, the combination of three characters "ffi", when occurring in sequence, are sometimes represented by a single glyph called a ligature, drawn in a slightly different manner than the characters occurring individually. As a third example, accented characters such as "c" are sometimes represented by more than one glyph (e.g., one for the character and one for the accent) and are sometimes represented by a single glyph (combining accent with character).

[0018] The unstructured document of some embodiments does not specify any relationship or association between the primitive elements, while in other embodiments it specifies a minimum amount of such relationships and associations. In some embodiments, the unstructured document may have some amount of structure but the structure is unrecognizable.
or not relied upon. In some embodiments the unstructured document has an unknown structure or is assumed to be unstructured.

[00119] Some embodiments generate, from the unstructured document, a structured document that includes associations and relationships between the primitive elements, groupings and orderings of the primitive elements, and properties of the groups of primitive elements. For instance, some embodiments use the primitive elements of the unstructured document to identify various geometric attributes of the unstructured document and use these identified geometric attributes (along with other attributes of the primitive elements) to define structural elements. Structural elements of some embodiments include associated primitive elements (e.g., words, paragraphs, joined graphs, etc.), guides, gutters, text flow, tables, etc. These structural elements are related in a hierarchical manner in some embodiments (e.g., a paragraph includes text lines, a text line includes words, and a word includes primitive glyphs). In some embodiments, the structured document serves two purposes - it identifies associated elements (e.g., the elements making up a table) and it identifies a flow order through the primitive elements (i.e., the order in which a human would be expected to read through the primitive elements in the document).

[00120] Upon receiving an unstructured document, some embodiments first parse the document into its constituent elements (e.g., primitive elements and their associated information such as coordinate locations, drawing order, etc.). For instance, a large block of text might be defined in the unstructured document as a number of character glyphs, each having x- and y-coordinates at which their anchors are placed on a particular page along with a scale factor determining the size of each glyph (and any other linear transforms that are to be applied), each glyph to be drawn on the page in a particular order (relevant to the compositing operation performed when one glyph overlays another). Some embodiments then perform geometric analysis on the primitive elements to define geometric attributes of the document. For example, some embodiments analyze the primitive elements to identify boundaries between primitive elements and regions bordered by the boundaries.

[00121] Figure 1 illustrates the overall flow of some embodiments. As shown, a document is initially (after parsing to identify the primitive elements, in some embodiments) analyzed by the geometric analysis modules 110. Geometric analysis modules 110 analyze a document to identify geometric attributes such as boundaries and regions bordered by the boundaries. In some
embodiments, the regions include zones that are bordered by primitive elements such as straight lines and narrow rectangles (i.e., particular primitive shapes and images).

[00122] Figure 2 illustrates a page 200 of an incoming document and various results from geometric analysis and document reconstruction. The incoming document is an unstructured document that has a collection of primitive elements that a human viewing the document would recognize as text, borders, a table, and a graphic object. Analysis result 205 illustrates that the geometric analysis modules 110 have recognized two zones Z1 240 and Z2 245 separated by boundaries 250 in document 200.

[00123] In some embodiments, the boundaries identified by geometric analysis modules 110 also include alignment guides. In some embodiments, an alignment guide is a vertical edge formed by the beginning or end of words (e.g., at the left edge of a column of left-aligned text). Similarly, in some embodiments, the regions identified by geometric analysis include gaps of unfilled white space between groups of glyphs (e.g., between guides). These gaps are called gutters in some embodiments.

[00124] Analysis result 210 illustrates a left-alignment guide 212 at the left edge of the first column of text and a gutter 214 spanning the white space between the two columns of text (for simplicity, the other guides and the columns of text arc not shown). As illustrated in Figure 1, the output of the semantic analysis modules 110 of some embodiments is zones 105, guides 115, and gutters 125.

[00125] The data output from geometric analysis modules 110 is sent to document reconstruction modules 120. Document reconstruction modules 120 continue the process of analyzing the unstructured document to define a structured document. In some embodiments, document reconstruction modules 120 create associations between primitive elements in order to define contiguous structural elements such as text, tables, and shapes. Some embodiments also define a hierarchy of the structural elements and relationships between the structural elements.

[00126] For instance, in some embodiments, the document reconstruction modules 120 create associations between glyphs, sets of glyphs, sets of sets of glyphs, etc. Some embodiments associate individual glyphs into words, words into text lines, text lines into paragraphs, etc. Analysis result 215 illustrates that individual lines 217 and paragraphs 219 are identified within
the first column of text.

[00127] The document reconstruction modules 120 also identify the layout of glyphs in order to define the text flow through the glyphs. Specifically, to define the text flow, some embodiments identify a reading order through the glyphs (or through the sets of glyphs), which represents the order in which a human would be expected to read through the glyphs on a page (e.g., from the bottom of a first column to the top of a second column, then skipping a separated text box in the center, etc.) Analysis result 220 illustrates that two columns are identified within the document 200 and that the reading flow 222 runs from the bottom of the first column to the top of the second column. In some embodiments, the identification and definition of layout and flow makes use of the zone results 205, the guide and gutter results 210, and the glyph association results 215.

[00128] The document reconstruction modules 120 also define other structural elements in a document that are associations between primitive elements other than glyphs or between structural elements. For instance, in some embodiments, document reconstruction modules 120 identify tables in a document as associations between regions identified by geometric analysis modules 110 as well as the glyphs and sets of glyphs within the regions. For example, some embodiments associate regions as cells of a table, and the glyphs inside each region as the table information. Analysis result 225 illustrates the identification of a table 227 with nine cells 229 in document 200 by document reconstruction modules 120. Some embodiments associate the primitive elements that form the table by defining a tabular structural element. Whereas in the initial document, what was viewed as a table was defined as an unassociated collection of primitive elements (lines and glyphs), after reconstruction the cells are identified in the tabular structural element as table cells and are individually or collectively editable. As further illustrated, in some embodiments, the table identification and reconstruction uses zone results 205, glyph association results 215, and layout and flow results 220.

[00129] Some embodiments also identify when two or more primitive graphic elements or graphic objects (e.g., shapes, images, photographs, bitmaps, etc.) in the document should be grouped as one structural graphic element. For instance, two objects that mostly overlap may be one element that is defined as two shapes or images in the unstructured document. The document reconstruction modules 120 join these two objects as one object. Analysis result 230 illustrates
that the two primitive shapes (a star and a hexagon) from the initial document 200 have been joined as one graphic 232 by the document reconstruction modules 120.

[00130] As illustrated in Figure 1, examples of the output of the document reconstruction modules 120 include semantic hierarchy data 135 (i.e., associations of glyphs), layout and flow data 145, table data 155, and joined graph data 165. Furthermore, in some embodiments, some of this information is also passed between the several document reconstruction modules 120. Figure 2 illustrates that all of this information is used to define a structured document 235. Structured document 235 has the same appearance as unstructured document 200, but the structured document 235 includes information about the structural elements and the associations, relationships, and hierarchy of elements, thereby enabling editing, more intuitive display, etc.

[00131] The data from the document reconstruction modules 120 (as well as, in some embodiments, data from the geometric analysis modules 110) is used by document display and interaction modules 130. Document display and interaction modules 130 enable a user to view, edit, scroll through, etc. a document. For example, sequence 140 illustrates a document displayed as two columns of text on a handheld device that is held upright. When the handheld device is rotated on its side, the text in the two columns is rearranged into three columns. This rearrangement cannot be done with an unstructured document, because it relics upon the associations between elements, especially the flow of text through glyphs that is not part of the unstructured document.

[00132] In some embodiments, document display and interaction modules 130 can also recognize a structural element (e.g., a paragraph, graphic object, etc.) that has been selected by a user and intelligently zoom to display the selected element. In some embodiments, the user selects a position of interest (i.e., a particular location in a displayed document), and the display and interaction modules 130 identify a qualifying structural element in the hierarchy of structural elements. Some embodiments define particular types of structural elements as qualifying structural elements. The qualifying structural element is used to define a region of interest that is highlighted in the display in some embodiments.

[00133] Sequence 150 illustrates a selection of a paragraph 170 (e.g., by a selection of a position of interest of interest within the paragraph) and the subsequent intelligent display of the paragraph and nearby text. Document display and interaction modules 130 also provide other
features such as intelligent selection of text and graphic objects, intelligent scrolling through a
document, etc.

[00134] Some embodiments use hierarchical profiling to modify how geometric analysis
and document reconstruction are performed on the fly, using intermediate analysis and
reconstruction results. Some embodiments check the intermediate results against profiles that
indicate what type of content a document includes and alter the reconstruction processes
accordingly. In some embodiments, the hierarchical profiles can instruct the analysis and
reconstruction modules to perform more or less processes, perform processes differently, or re-
perform processes. For instance, if intermediate analysis results indicate that a document is one
page long, has one column of text, and no shapes or images, then some embodiments will only
perform processes to associate the glyphs into words, lines, and paragraphs. Table identification,
for instance, will not be performed.

[00135] Some embodiments employ various novel efficiency techniques for more efficient
memory and processing usage. For instance, some embodiments perform some of the above
described processes by using cluster analysis, which is a technique used to identify groups of
elements that are closely spaced in some way relative to other elements. Some embodiments use
cluster analysis to identify guides based on numerous words starting at, ending at, centered on or
otherwise aligned with the same or nearly the same x-coordinate. Some embodiments use cluster
analysis to recognize different size gaps between glyphs so as to identify gaps between words
and gaps larger than those between words. Some embodiments also use cluster analysis to
identify primitive graphics (e.g., shapes, images) that should be joined into single graphics.

[00136] Some embodiments perform cluster analysis efficiently by using ordered data
(e.g., primitive element position data) that references unsorted data, and by storing partitions of
the data using a single value. A partition, as this term is used in the present invention, divides a
sequence, or linearly ordered set, into subsequences, which are subsets of the sequence with the
same order relation. Furthermore, a partition has the properties that (i) every member of the
original sequence is contained in exactly one of the partition's subsequences, and (ii) given two
of the partition's subsequences S and T, either all the members of S are less than all the members
of T or all the members of T are less than all the members of S, according to the order relation.

Storing a partition as a single value enables various cluster analysis functions, such as examining
multiple partitions, to be performed more efficiently in some embodiments.

[00137] Some embodiments also gain efficiency in the document reconstruction process by using an application programming interface (API) that minimizes the amount of copying of data while appearing to the user of the API (e.g., a programmer or a software application using the API) as though the data is freely modifiable. Some embodiments store data in a randomly ordered array, then define a sorted array of references to the data and share this sorted array among numerous collection objects (e.g., character sequence objects, which are collections of character data) to optimize the usage of memory and processing. Both of these efficiency enhancements, as well as others, are used in some embodiments to enable document reconstruction to be performed on a limited-resource device, such as a cell phone, media player, etc. (e.g., an iPhone®).

[00138] Although the above-described overview of some embodiments was provided by reference to the examples illustrated in Figures 1 and 2, one of ordinary skill will realize that these examples were meant only as exemplary embodiments that introduced the features and operations of some embodiments of the invention. One of ordinary skill will realize that many embodiments have features and operations that are different than those illustrated in Figures 1 and 2. For instance, although geometric analysis has been described as one set of modules 110, one of ordinary skill would recognize that some embodiments do not necessarily identify all geometric attributes at once. For example, some embodiments do a subset of geometric analysis first (e.g., region analysis to identify one or more zones in the document) and then guides and gutters are identified on a zone-by-zone basis.

[00139] More detailed examples of some embodiments will be described below. Section II describes the identification of regions (i.e., zones) of a document based on boundary primitive elements and the definition of a hierarchical structure (e.g., a document object model) that forms the framework of a structured document. Section IV then describes the identification of boundary elements for glyphs (e.g., alignment guides) and particular empty spaces between alignment points (gutters). Next, Section IV details the creation of associations between glyphs and sets of glyphs to define structural elements such as words, text lines, paragraphs, columns, etc., as well as the definition of a flow order through these structural elements (as well as other elements such as graphics, tables, etc.). Section V describes the identification of tables from primitive elements
and the creation of associations between the primitive elements to form a tabular structural element, while Section VI describes the identification of primitive graphic elements that should be grouped together and the creation of associations between such primitive elements to define compound graphic elements. Next, Section VII describes the modification of content processing (using the example of document reconstruction as detailed in Sections H-VI) based on matching intermediate processing results to a set of predefined profiles, which can improve the efficiency and accuracy of the content processing.

[00140] Next, Section VIU details various aspects of using a structured document (e.g., a document defined by a document object model) to display the document on a small-screen device (e.g., a handheld phone or media player). Both the adaptation of the document display to the small screen and the display of an identified region of interest are discussed. Section IX describes the use of the structured document to define a selection of text (or other elements of the document) in response to user input.

[00141] Section X then describes various methods for improving the efficiency of cluster analysis techniques, which (among other uses) are used for identification of alignment guides, words and glyph spacing, and compound graphics in the document reconstruction process. Next, Section XI details methods and data structures that enable more efficient parsing and analysis of a document. These data structures illustrate one manner of creating associations between glyphs (e.g., to form words, text lines, paragraphs, etc.) that can be used in the document reconstruction process. However, one of ordinary skill in the art will recognize that many other ways of creating associations between primitive elements (e.g., glyphs, graphic elements, etc.) to define structural elements (e.g., paragraphs, tables, compound graphics, etc.) are possible, as is well known in the art. Next, Section XII describes the software architecture of a document reconstruction application of some embodiments, and Section XIII describes a computer system that implements some embodiments of the invention.

II. ZONE ANALYSIS

[00142] When there are multiple articles, sections or categories of information on a page, these are often delineated by lines, images or shapes. Although a human can easily identify the manner in which graphical cues are intended to indicate how the page is broken up into zones, this is a nontrivial problem for a computer (particularly in the presence of a mixture of uranich
primitive elements, some of which are intended as page content while others are intended to
delineate content zones).

Some embodiments of the invention provide methods for identifying boundaries and the regions bordered by those boundaries (e.g., zones) based on the primitive elements (e.g.,
the shapes and images) of an unstructured document. In some embodiments, the regions are used in subsequent reconstruction of the document as well as for compartmentalization of further
reconstruction processes. Some embodiments generate a region graph (i.e., hierarchical structure such as a tree) that is populated with content and enables the association of content with the
region in which the content is located. Some embodiments perform the region identification on a
page-by-page basis.

Figure 3 conceptually illustrates a process 300 for identifying zones of a page of a
document and generating a zone tree for the page in some embodiments. Process 300 will be
described in conjunction with Figure 4. Figure 4 illustrates the sequence of identifying zones of a
page 400 of a document and generating a zone tree 430 for the page according to some
embodiments. As shown in Figure 3, process 300 begins by receiving (at 305) a page of a
document. In some cases a document includes numerous pages (e.g., an e-book), whereas in
other cases a document will only be one page (e.g., an advertisement flyer).

Next, the process identifies (at 310) zones on the page. In some embodiments, the
identification of zones includes identifying zone borders and intersections and then traversing
the zone borders to identify the zones. Referring to the example of Figure 4, process 300
identifies that page 400 includes five zones: zones A 405, B 410, C 415, D 420, and E 425.

After identifying the zones, process 300 generates (at 315) a zone graph (i.e.,
hierarchical structure such as a tree) for the page. The zone graph illustrates the hierarchy of the
zones. For instance, zone tree 430 illustrates that a zone for the page (node P) includes four
zones A, B, C, and D. Furthermore, zone D includes zone E, as zone E is fully within zone D. In
some embodiments, a first zone is the parent of a second zone when the second zone is wholly
within the first zone. A parent and a child can share one or more borders in some embodiments.

After generating the zone graph, process 300 inserts (at 320) the content of the
page into the zone graph. The process then ends. In some embodiments, a page includes text,
graphics, or other content. Each particular content grouping (e.g., an image, paragraph, column, etc.) is placed as a child of the smallest zone that fully contains the particular content grouping. In some embodiments, the insertion of content objects into the zone graph is performed later in the document reconstruction process, once the content has been further analyzed (e.g., grouping text into paragraphs, identifying tables, etc.). Furthermore, as document reconstruction is performed, some embodiments update the zone graph with content subtrees for each zone.

A. Terminology

[00148] Figure 5 illustrates a page 500 of a document that includes several zones. Page 500 includes numerous zone borders, including zone borders 505-509. Zone borders, in some embodiments, are horizontal or vertical (i.e., rectilinear) strips with a thickness defined by the zone border graphics that contribute to the zone border. The thickness of a zone border, in some embodiments, is the width, in its narrow direction, of an upright bounding box of the zone border graphics that contribute to the zone border. In some embodiments, an upright bounding box for a particular element or set of elements is the smallest upright rectangle (in the coordinate system being analyzed) that fully envelops the element or set of elements.

[00149] Zone border graphics are graphic objects (e.g., shapes, images, lines) on a page that either arc narrow rectangles or have an upright bounding box that is a narrow rectangle. For instance, zone borders 505-509 are all lines with a particular (relatively narrow) thickness. In some embodiments, zone border graphics include relatively narrow objects, all or part of the rendering of which fills all or part of a zone border. In some embodiments, zone border graphics also include objects whose boundary contributes to a zone border (e.g., one side of a filled polygon can indicate all or part of a zone border even though the polygon itself is not narrow and does not fit in the border bounds).

[00150] Zone borders graphics, however, need not be perfectly straight lines or perfectly rectilinear. For instance, Figure 6 illustrates a page 600 that includes zone border graphics 605. Zone border graphics 605 are not perfectly vertical strips: instead they are images of twigs that are aligned very close to vertically. Some embodiments will recognize the graphic as a zone border graphic, whereas some embodiments will not.

[00151] Page 500 of Figure 5 also includes numerous zone border intersections, such as
intersections 510 and 511. In some embodiments, a zone border intersection is a rectangular
intersection of a horizontal zone border with a vertical zone border. As intersection 511 illustrates, a zone border intersection need not be at the end of a zone border. Zone border intersections in the middle of a zone border break the zone border into one or more zone border intervals, in some embodiments. For instance, the bottom zone border of page 500 is broken into zone border intervals 515, 516, 517, and 518.

[00152] A zone, therefore, is a closed region bounded by a collection of zone border intervals that form an upright rectilinear shape in some embodiments. Upright rectilinear shapes are any polygons that can be formed by horizontal and vertical line segments, including but not limited to upright rectangles, which are rectangles formed from horizontal and vertical line segments. Each zone has an upright rectilinear outer bound which is a shape formed from the outer sides of its zone border bounding rectangles. Each zone also has an upright rectilinear inner bound, which is a shape formed from the inner sides of its zone border bounding rectangles.

[00153] Page 500 includes zones P 526 (the page bounds), A 520 (an arch-shaped zone that includes the thin strips on the left and right side as well as the area above zones C and D), B 521, C 522 (the left zone that shares borders with zone E), D 523 (the right zone that is a mirror image of zone C), E 524, and G 525. Zones have outer bounds and inner bounds in some embodiments, defined by the outer and inner sides of the zone borders.

[00154] Figure 7 illustrates a zone tree 700 for page 500, with zone P (the page borders) a parent of zones A, C, E, and D; zone B a child of zone A; and zone G a child of zone D. Zones B, E and G are examples of islands. An island is a zone that does not share a border interval with its parent zone. Although zone E shares its border intervals with zones C and D, because neither of those zones actually encloses zone E, neither of them is a parent of zone E. The zone tree also illustrates that the nodes have been populated by the content that they include. In some embodiments, the portion of a document object model (DOM) for each page is built on the nodes of the zone tree of the page. A document object model is a representation of a document as a graph whose nodes are objects. In some embodiments, this graph is a tree, its leaf nodes represent primitive elements, and its non-leaf nodes are structure objects that express the relationships between their child nodes as well as the properties that those child nodes have as a
group. In some embodiments, the order of the children of a node represents the reading order of those children. In some embodiments, the root node is a document node, its children are page nodes, the zone tree descends from each page node, a flow tree (including nodes representing structures such as tables, text boxes, layouts, columns, paragraphs, lists, and text lines) descends from some of the zone nodes, and nodes representing primitive elements (such as glyphs, shapes and images) are the children of some of the nodes in the flow tree. In some embodiments the structure nodes include properties that express relationships between nodes in addition to the relationships expressed by the tree's parent-child relationships (its directed graph edges). For example, the paragraph that states a new column may be a continuation of the paragraph that ends a previous column, without a paragraph break between the two. In this case, there would be two paragraph nodes in the tree, each with a different column node parent, but they would have properties pointing to one another to indicate that they are two nodes representing parts of a single, common paragraph. A DOM, in some embodiments, is a hierarchical representation of a document that includes all the structural elements of the document. Some embodiments define content to be a child of a particular zone when the content is located entirely inside the outer bound of a particular zone and is not located entirely inside the outer bound of any child of the particular zone. As such, zone B includes header text, zones C and D include standard text, and zones E and G include images.

B. Rotation Groups

Some embodiments define several rotation groups on a page and analyze the zones and content of each rotation group separately. In some embodiments, rotation groups are similar to zones except that they do not have any zone borders. Instead, a rotation group is defined to include all content that is rotated by the same angle (or nearly the same angle to within a particular threshold that is sufficiently small as to be difficult for a human viewer to distinguish). Figure 8 conceptually illustrates a process 800 of some embodiments for defining rotation groups on a page. As shown, process 800 receives (at 805) a page of a document. In some cases, the page is the only page of the document, whereas in other cases the page is one of multiple pages. Some embodiments perform rotation group analysis for a multi-page document (or a multi-page section) all at once, rather than page-by-page.

The process then determines (at 810) the rotation angle of each object on a page.
In some embodiments, irregularly-shaped images are assumed to have a rotation angle of zero. For instance, the image in zone E of page 500 is irregularly shaped, and would not be given a non-zero rotation angle. Horizontally-aligned text also has a rotation angle of zero, while text that is aligned off the x-axis is given a rotation angle. For example, the text in region F 530 of page 500 would have a rotation angle of approximately -45 degrees. Similarly, the text 610 ("Organic" and "Pure") in page 600 would have its own rotation angle. In embodiments that also place graphic objects into rotation groups, the rectangular image 615 above text 610 would have the same rotation angle as text 610.

Next, process 800 orders (at 815) the objects by rotation angle. The process then groups (at 820) the objects into clusters with a spread in rotation angle that is below a particular threshold. In some embodiments, the spread that is compared to the particular threshold is the smallest rotation angle in the group subtracted from the largest rotation angle in the group. The use of a non-zero threshold allows the grouping to account for minor errors in the content definition in the initially received document (e.g., a line of text that is very slightly off of horizontal).

Process 800 then analyzes (at 825) each rotation group separately. The process then ends. On most pages, most of the analysis will involve the upright (zero angle) group. Some embodiments do not perform zone analysis for groups other than the upright group, and instead simply classify the content of the rotated groups as children of the page as a whole. In some embodiments, each rotation group has a coordinate system in which its content appears upright. In such embodiments, each rotation group has its own zone tree with content that fits into the DOM for the document. Some embodiments define one rotation group for each distinguishable angle by which content on the page is rotated. The analysis on each group is described in detail below.

C. Identifying Zone Borders and Intersections

Figure 9 conceptually illustrates a process 900 of some embodiments for identifying zone borders and intersections. Process 900 will be described in conjunction with Figure 10. Figure 10 illustrates a page 1000 that includes various graphics and text.

As shown in Figure 9, the process receives (at 900) a rotation group and
normalizes the group to an upright coordinate system. In some embodiments, normalizing the group to an upright coordinate system involves defining a coordinate system for the group such that all objects in the group are vertical or horizontal (e.g., text lines are horizontal in the coordinate system). The following discussion assumes that the rotation group is the upright (zero-angle) group. One of ordinary skill in the art would be able to apply the same techniques to rotation groups with non-zero angles in a coordinate system in which their content appears upright. Some embodiments remove content from other rotation groups before performing zone identification for a particular rotation group. For instance, some embodiments would remove text 610 and image 615 from page 600 in Figure 6 before performing zone identification and analysis in the upright rectilinear coordinate system.

[00161] The process then identifies (at 910) potential zone borders. Potential zone borders, in some embodiments, include any horizontal or vertical graphic object that is sufficiently narrow. The determination of whether a particular graphic object is sufficiently narrow uses an absolute measure (e.g., when the smaller dimension of the upright bounding rectangle of the graphic object is less than 1/24 of inch) in some embodiments. In other embodiments, the determination uses a relative measure (e.g., the larger dimension of the upright bounding rectangle is eight times the size of the smaller dimension), or a combination of absolute and relative measures (e.g., the narrow dimension could be allowed to be up to 1/12 of an inch, but the relative measure of 8:1 applies). Some embodiments adjust the threshold in relation to the size of the page. For instance, the above examples might apply to a standard 8.5 x 11 inch page, whereas a much larger page could have larger potential zone borders.

[00162] Referring to Figure 10, page 1000 includes several lines that would be classified as potential zone borders: horizontal borders 1005-1010 and vertical borders (101 1-1016). However, graphic object 1020 would generally not be considered a potential zone border, because it is too thick in the x-direction.

[00163] Some embodiments also identify all upright rectilinear shapes that have at least a threshold size and use the sides of these shapes as potential zone borders. In some embodiments, the threshold size is a particular area, whereas in other embodiments a threshold width and a threshold height must be surpassed. For instance, object 1020 might have an area large enough to qualify its edges as potential zone borders, but it is too narrow to be a separate zone. Star object
1025. on the other hand, is not an upright rectilinear shape and as such its edges would not qualify as a zone border. As such, these objects would simply be classified as content (specifically, graphic objects) that are within one zone or another. Some embodiments set the bounds of each potential zone border identified as the side of an upright rectilinear shape as the upright rectangle bounding the side, including the stroke width if stroked. Some embodiments also include the page borders as zone borders if they are upright rectilinear in the coordinate system of the rotation group.

[00164] After identifying potential zone borders, process 900 removes (at 915) borders or portions of borders that intersect with other objects on the page. For instance, potential border 1015 is obscured by star object 1025, and as such would be broken into two potential zone borders (the area above the star and the area below the star). Some embodiments also remove zone borders that intersect character bounding boxes. A character bounding box for a particular character, in some embodiments, is the smallest rectangle that completely encloses the character.

For instance, potential zone border 1010 crosses the characters "Lorem Ipsum". As such, some embodiments would remove potential zone border 1010 from consideration.

[00165] Next, process 900 merges (at 920) borders. Some embodiments merge borders that are parallel and either overlapping or close to overlapping. Borders overlap when their bounds intersect. For instance, when two very narrow rectangles of different width are drawn such that one completely envelops the other, the two potential zone borders would be merged.

Some embodiments slightly expand the bounds (both in width and length of the potential zone borders to test for overlap. Accordingly, borders 1013 and 1014 in Figure 10 would be merged into one zone border 1027, with a thickness greater than that of borders 1013 and 1014.

[00166] Process 900 then determines (at 923) whether any merged borders remain unprocessed. When no borders were merged, or all merged borders have been processed, the process proceeds to 945. described below. Otherwise, the process selects (at 925) an unprocessed merged border. The process then determines (at 930) whether the merged border is too thick or includes too many zone border graphics. A merged border is too thick, in some embodiments, when its width in the narrow direction is above a particular threshold. In some embodiments, the test for thickness is the same as whether a graphic object is narrow enough to be classified as a zone border initially. When the process determines that the border is not too thick, the process
proceeds to 923 which is described above. Otherwise, when the merged border is too thick, the
process removes (at 935) the merged border from the potential zone border candidates and
classifies it as a single graphic object, then proceeds to 923. For instance, this could happen
when an image is drawn as a series of narrow rectangles or a bar graph is drawn with narrow and
closely spaced bars.

[00167] Once all merged borders are examined, the process identifies (at 945) zone border
intersections. As discussed above, zone border intersections are identified wherever a horizontal
border intersects a vertical border. Some embodiments also identify near-intersections and
classify these as intersections. To find near-intersections, borders are extended a small amount
and then tested for intersection. Some embodiments extend the borders a fixed amount (e.g., one-
fourth of an inch), while other embodiments extend each border an amount that is a percentage
of the length of the particular zone border. When the lengthened borders intersect, the near-
intersection is classified as an intersection and the two borders are extended to fully cross the
thickness of the other. As an example, borders 1027 and 1008 in Figure 10 do not quite
intersect. However, they are close enough that they would be classified as intersecting and are
extended such that they intersect.

[00168] The process then eliminates (at 950) borders with less than two intersections.
Once a border is removed, any borders that intersected the removed border must be removed
to determine whether they still have at least two intersections. In the example page 1000, border
1006 and the two remaining portions of border 1015 would be removed, as they have no zone
border intersections. Once the zone borders and intersections are identified, the process trims (at
955) the zone borders to remove any portions extending past the outermost intersections. For
instance, the borders 1027 and 1009 extend past their intersection. These would be trimmed to
extend only to the outermost bound of each other. After trimming the borders, the process stores
(at 960) the zone border and intersection information for future use (e.g., in identifying zones).
The process then ends.

[00169] At this point, the zone border intervals and zone border intersections have all been
determined. Figure 11 illustrates vertical zone border intervals 1105, 1115, 1125, 1135, 1145,
1155, 1165, and 1175 as well as horizontal zone border intervals 1110, 1120, 1130, 1140, 1150,
1160, 1170, and 1180. Figure 11 also illustrates zone border intersections 1102, 1112, 1113,
112, 1123, 1132, 1133, 1142, 1143, 1152, 1162, 1172, 1182, and 1192.

D. Identifying Zones

[00170] Once the zone borders and zone border intersections are identified, the zones can be identified. **Figure 12** conceptually illustrates a process 1200 of some embodiments for identifying zones. Process 1200 will be described in conjunction with **Figures 13** and **14**. **Figures 13** and **14** illustrate the application of process 1200 to identify the zones of page 1000. Each of the figures is illustrated as a sequence. **Figure 13** illustrates a sequence 1305-1330 to identify a first zone border. Arrows in **Figure 13** illustrate direction vectors and dashed lines illustrate a path taken through the zone border intervals to define a zone. **Figure 14** illustrates the zones identified by process 1200.

[00171] As shown in **Figure 12**, process 1200 receives (at 1205) zone borders and intersections for a group or page. In some embodiments, the zone borders and intersections are the output of process 900 described above. The process then determines (at 1207) whether there are any zone border intervals. When there are none, the process ends. Otherwise, the process assigns (at 1210) two direction vectors to each zone border interval (i.e., horizontal intervals have vectors pointing right and left, and vertical intervals have vectors pointing up and down). **Figure 13** illustrates (at 1305) that each of the border intervals for page 1000 starts with direction vectors in both directions.

[00172] Next, the process selects (at 1215) a border interval b, an intersection i, and a direction d. Some embodiments select the starting point randomly, whereas other embodiments use a heuristic such as the top- and left-most intersection in a particular direction. **Figure 13** illustrates (at 1305) a random selection of starting at intersection 1182 moving upwards along interval 1115. Process 1200 then proceeds (at 1220) in the direction d from intersection i until arriving at the next intersection.

[00173] Once the intersection is reached, the process determines (at 1225) whether the intersection is the starting intersection selected at 1215. When the intersection is the original starting intersection, the process proceeds to 1265 which is described below. Otherwise, the process determines (at 1230) whether the path through the zone border intervals can turn clockwise at the intersection. When the path can turn clockwise, the path docs so (at 1235). The
process then proceeds to 1255 which is described below. When the path cannot turn clockwise, the process determines (at 1240) whether the path can continue straight through the intersection. When the path can continue straight, then the path docs so (at 1245). The process then proceeds to 1255 which is described below. When the path cannot continue straight, the path turns (at 1250) counterclockwise to the next border interval. By the choices made in steps 1230 and 1240, the process 1200 exhibits a preference for a clockwise turn at each border intersection. Some embodiments will instead exhibit a preference for counterclockwise turns, which gives the same results.

100 174] The process sets (at 1255) the new border interval as the current border interval \( b \), and the new intersection as the current intersection \( i \). The process then sets (at 1260) the direction \( d \) moving away from intersection \( i \) along border \( b \). The process then proceeds to 1220 which was described above.

100 175J Once the original intersection is reached, process 1200 defines (at 1265) a zone \( Z \) as the set of border intervals traversed since operation 1215. As noted above, Figure 13 illustrates the traversal of a set of zone border intervals according to process 1200. At 1305, after selecting interval 1145 moving up from intersection 1182 to start (shown by the circle and short arrow in the figure), the path comes to intersection 1112. Turning clockwise is an option, so the path turns (at 1310) to interval 1120, then clockwise again at intersection 1122 to interval 1155. The path turns (at 1315) clockwise yet again at intersection 1132 to interval 1150, but then at intersection 1142 cannot either turn clockwise or continue straight through. Instead, the path turns counterclockwise to interval 1145, then again at intersection 1152 to interval 1160 to proceed towards intersection 1162. At intersection 1162, the path turns (at 1320) clockwise to interval 1175, then clockwise again at intersection 1172 to interval 1180. Interval 1180 returns to the path to the original intersection 1182.

[00176] Figure 13 illustrates (at 1325) the zone 1300 defined by the traversal of intervals 1115, 1120, 1155, 1150, 1145, 1160, 1175, and 1180. as well as the direction vectors used in that traversal. Returning to process 1200, after defining (at 1265) the zone \( Z \), the process removes (at 1270) the direction vectors used to traverse zone \( Z \). Figure 13 illustrates (at 1330) the zone border intervals of page 1000 with the direction vectors used to traverse zone 1300 removed.

[00177] Process 1200 next removes (at 1275) all hrnrir intervals with no remaining
direction vectors. This will not occur after the first zone is identified, but can happen after any of
the further zones are identified. When the zone Z is an island (i.e., a zone that shares no borders
with its parent), process 1200 classifies (at 1280) the zone as such. In embodiments in which the
preference is for clockwise turns, then a zone defined by traversing its center in a
counterclockwise direction will be an island.

[00178] The process then determines (at 1285) whether any zone border intervals remain.
When more zone border intervals remain, the process proceeds to 1215 which was described
above. Otherwise, once all zone border intervals are used in both directions, the process has
defined all the zones for the page. The process then stores (at 1290) the zone information. The
process then ends.

[00179] Figure 14 illustrates the continuation of the process 1200 applied to page 1000.
For simplicity, Figure 14 does not illustrate every move through the traversal of the zone border
intervals. First, starting at any of the intersections 1113, 1123, 1133, and 1143, the two zones
1435 and 1440 arc identified. These zones are duplicates of each other, as will occur in the case
of islands that have no non-island children. Some embodiments remove duplicate zones. Other
embodiments, however, treat the zones as two: one that is a regular zone, and the other that is an
island. Next, starting at intersection 1192 results in zone 1445 (the page borders), because all
possible turns off of the page borders would be counterclockwise moves. Finally, this leaves
zones 1450 and 1455, which are traversed and removed. Once all the zones are traversed, there
arc no remaining zone border intervals.

E. Generating the Zone Tree

[00180] Once the zones have been identified, the zone graph (zone tree) can be generated.
The zone tree is used, in some embodiments, in document reconstruction that is done on a zone-
by-zone basis. Figure 15 conceptually illustrates a process 1500 of some embodiments for
generating a zone tree. As shown, the process receives (at 1505) zones and content objects. In
some embodiments, these zones have been identified by a process such as process 1200. The
process then sorts (at 1510) the zones by area. Some embodiments treat an island as larger than a
non-island when their areas are equal for the purposes of sorting the zones.

[00181] Next, the process selects (at 1515) the smallest zone as z. The process then
determines (at 1520) whether zone z has a node yet in the zone graph for the page. When z has a
node, the process proceeds to 1530 which is described below. Otherwise, when z does not yet
have a node, the process 1500 defines (at 1525) a node for zone z.

[00182] Next, the process selects (at 1530) the next smallest zone as zone p. The process
then determines (at 1535) whether zone p contains zone z (i.e., whether the outer bounds of zone
τ are completely within the outer bounds of zone p). When zone p contains zone z, the process
determines (at 1540) that zone z is a child of zone p. Based on this, the process defines (at 1545)
a node for zone z in the node graph. The process then defines (at 1550) an edge from zone p to
zone z. The process then proceeds to 1565 which is described below.

[00183] When, at 1535, the process determines that zone p does not contain zone z, the
process determines (at 1555) whether there are any zones larger than the current zone p. When
there are larger zones remaining, the process proceeds to 1530 and selects the next smallest zone
as zone p to test whether the new zone p is a parent of zone z. Otherwise, when there are no
zones larger than zone z?, the process determines (at 1560) that zone z has no parent zones.

[00184] Next, the process determines (at 1565) whether there are any zones larger than
zone z. When there are larger zones, the process removes (at 1570) zone z from the set of zones
from which to select and proceeds to 1515 to select another zone for parent-child analysis.

[00185] Figure 16 illustrates the zones 1435 (A), 1440 (A’), 1455 (B), 1450 (C), 1300 (D)
and 1445 (E) of page 1000 (shown in Figure 10) sorted in size order (A’ is the island for A) and
placed into node graph 1600. Using process 1500, first a node for zone A (the smallest zone)
would be defined, then the zones would be tested until the process determined that island zone
A’ was a parent of zone A, at which point zone A would be defined in the node graph, and an
edge from A’ to A would be defined. Next, zone D would be determined to be the parent of
island zone A’, and then zones B, C, and D would all be determined to be children of island zone
E, which has no parents. In some embodiments, levels of zones and island zones always alternate
in the zone graph. Thus, islands E and A’ are at the first and third level of graph 1600, and zones
B, C, D, and A are at the second and fourth level.

[00186] Once all zones have been analyzed, the process proceeds to 1573 and determines
whether there are any unprocessed content objects. When there are no content objects (i.e., the
document is blank except for zone borders), or all content objects have been processed, the process proceeds to 1597, described below. Otherwise, the process proceeds to 1575 and selects a content object \( c \). The process then defines (at 1580) a node for the object \( c \). A content object, in some embodiments, is a primitive object (e.g., a glyph, shape or image). The process then determines (at 1585) the smallest zone \( z \) that contains content object \( c \). Once the zone \( z \) containing content object \( c \) is determined, the process defines (at 1590) an edge in the zone graph from zone \( z \) to content object \( c \). When all objects have been added, the process stores (at 1597) the zone graph. The process then ends.

[00187] In some embodiments, the content in each zone is further analyzed (e.g., grouping text into paragraphs, identifying tables, etc.). Furthermore, as document reconstruction is performed, some embodiments update the zone graph with content subtrees for each zone, where those content subtrees include structure nodes that represent the hierarchical grouping of the primitive objects of the zone. By performing zone analysis first, one ensures that content from different zones is not inappropriately grouped in the subsequent document reconstruction steps.

[00188] In some embodiments, the identification of geometric attributes such as boundaries and the regions bordered by those boundaries (e.g., zones) sets the stage for further document reconstruction. For example, profiles may depend on zone geometry and structure elements such as tables or text boxes may be recognized from the zone geometry.

F. Software Architecture

[00189] In some embodiments, the zone analysis processes described above are implemented as software running on a particular machine, such as a computer, a media player, a cell phone (e.g., an iPhone®), or other handheld or resource-limited devices (or stored in a computer readable medium). Figure 17 conceptually illustrates the software architecture of a zone analysis application 1700 of some embodiments for performing zone analysis on a document. In some embodiments, the application is a stand-alone application or is integrated into another application (e.g., a document reconstruction application), while in other embodiments the application might be implemented within an operating system.

[00190] Zone analysis application 1700 includes a border identification module 1705, an interval and intersection identification module 1710, a zone identification module 1715, and a
zone graph builder 1720, as well as zone information storage 1725.

[00191] Figure 17 also illustrates document content 1730. Border identification module 1705 receives information from the document content 1730. In some embodiments, this information is information about all of the graphics (e.g., shapes, images, lines, etc.) in the document. The border identification module 1705 identifies potential zone borders and passes this information to the interval and intersection identification module 1710, as well as to the zone information storage 1725. In some embodiments, border identification module 1705 performs some or all of process 900.

[00192] The interval and intersection identification module 1710 receives zone border information from the border identification module 1705 and/or the zone information storage 1725. The interval and intersection identification module 1710 identifies zone border intersections and zone border intervals based on the potential zone borders identified by module 1705. The identified zone border intersections and zone border intervals are passed to the zone identification module 1715 as well as storing in zone information storage 1725. In some embodiments, interval and intersection module identification 1710 performs some or all of process 900.

[00193] The zone identification module 1715 receives zone border information from the border identification module 1705, zone border intersection and zone border interval information from the interval and intersection identification module 1710, and/or information from the zone information storage 1725. Zone identification module 1715 identifies zones based on the information from modules 1705 and 1715. The identified zones are passed to the zone graph builder as well as storing in the zone information storage 1725. In some embodiments, zone identification module 1715 performs some or all of process 1200.

[00194] The zone graph builder 1720 module receives zone information from the zone identification module 1715 and/or the zone information storage 1725, as well as content information from the document content 1730. Zone graph builder 1720 defines the zone graph for a document based on the zone information, and populates the zone graph with content information. In some embodiments, the zone graph builder 1720 populates the zone graph as content information is identified by other reconstruction processes, such as those described in the Sections below. In some embodiments, zone graph builder 1720 performs some or all of process
In some embodiments, the results of the processes performed by the above-described modules or other modules are stored in an electronic storage (e.g., as part of a document object model). The document object model can then be used for displaying the document on an electronic display device (e.g., a handheld device, computer screen, etc.) such that a user can review and/or interact with the document (e.g., via touchscreen, cursor control device, etc.).

III. GUIDE AND GUTTER ANALYSIS

Some embodiments of the invention provide methods for identifying geometric attributes such as boundaries (e.g., alignment guides) and unfilled space (e.g., gaps of unfilled white space between groups of glyphs, called gutters) in a document or portion of a document. In some embodiments, a gutter is the white space between two alignment points (e.g., between a right-alignment point and a left-alignment point). Identification of guides and gutters is used in subsequent reconstruction procedures, such as column identification and splitting of text lines, in some embodiments. Some embodiments identify guides and gutters on a zone-by-zone or page-by-page basis.

Figure 18 illustrates an overall process 1800 of some embodiments for identifying guides and gutters in a document. Process 1800 will be described in conjunction with Figure 19, which illustrates a page 1900 having two columns of text, and the guides and gutters identified on page 1900. As shown in Figure 18, process 1800 receives (at 1805) a portion of a document. This portion may be multiple pages, a page, or a zone that has been identified by prior zone analysis. The portion of document may include words that have been reconstructed from glyph primitives by methods described elsewhere in this application.

The process then applies (at 1810) cluster analysis to determine guides of the received document portion. Cluster analysis enables the process to determine x-coordinates where the ends or beginnings of words are grouped together, making those x-coordinates likely alignment guides. As mentioned, Figure 19 illustrates a page 1900 with two columns of text. Page 1900 includes as set of guides 1905. Some embodiments determine bottom and top lines of columns as guides, whereas other embodiments only determine left- and right-alignment guides.
Some embodiments also identify guides for other alignments, such as center alignment or the alignment of decimal points in listings of numbers. Cluster analysis and the guide determination process are described in further detail below.

[00199] Next, the process determines (at 1815) the gutters of the document portion. Some embodiments use information from operation 1810 to determine the gutters. Figure 19 illustrates a gutter 1910 that is determined for page 1900 between the right-alignment guide of column one and the left-alignment guide of column two. Some embodiments treat the page margins as gutters, while other embodiments do not. Once the guides and gutters are determined, the process 1800 uses (at 1820) the guides and gutters for further reconstruction of the document. The process then ends.

A. Density Clustering

[00200] Some embodiments determine right- and left-alignment guides by searching for text lines that start or end at the same or nearly the same x-coordinate on a page and determining whether sufficient evidence exists that the x-coordinate is actually an alignment point. Some embodiments use a form of cluster analysis called density clustering to determine alignment guides. The density clustering of some embodiments takes advantage of the memory and processing efficiencies described below in Section X so that it can be performed on a resource-limited device (e.g., an iPhoncfl*).

[00201] Density clustering is often applicable to problems in which there is a substantial amount of "noise" or random data mixed in with otherwise clearly visible clusters. When the data is a set of real numbers, the clusters are identified as subsets that optimally meet given density constraints. The constraints are generally designed to pick out subsets that are relatively denser than others. For instance, some embodiments use a minimum size of a cluster and a maximum spread of a cluster as constraints.

[00202] Figure 20 conceptually illustrates a process 2000 of some embodiments for performing density clustering. As shown, the process receives (at 2005) a set of input data. In some embodiments, the input data is coordinate data of character glyphs on a page. For example, in using density clustering to find left-alignment guides, the input data is the x-coordinate of the anchor of the first letter of each word on the page.
The process then sorts (at 2010) the set of input data. Some embodiments sort the data in ascending order, while other embodiments sort the data in descending order. For instance, in the case of using density clustering to determine alignment guides, the data (x-coordinate values) is sorted from lowest to highest x-coordinate value such that if two x-coordinate values are equal they are next to each other in the sorted data (unless there are other words with the same x-coordinate value that fall in-between the two). Some embodiments create a new array for the sorted data, while some embodiments use an indirectly sorted array of indices as described below in Section X.

Next, process 2000 determines (at 2012) whether the set has at least two pieces of data. If not, then the process ends, as there is nothing to cluster. Otherwise, the process proceeds to determine (at 2015) the set of differences between subsequent data in the sorted set. Such a set will have one less value than the set of input data. As an example, when there are three words on a page, the two values in the set of differences are the difference between the x-coordinate values of the first and second words and the difference between the x-coordinate values of the second and third words.

Next, the process sets (at 2020) a variable $d$ to the largest uncvalued difference in the set of differences. For instance, when the differences for a set of words are 0.7 inches, 0.2 inches, 0.0 inches, and 0.4 inches, then the variable $d$ would initially be set to 0.7 inches. The process then partitions (at 2025) the sorted data wherever the difference is greater than or equal to $d$ to generate a set of subsets of the data. The first partition will always partition the sorted data only at differences equal to $d$, because $d$ will be set to the largest difference. In the above example of five data values with differences of 0.7, 0.2, 0.0, and 0.4, the partitioning would generate two subsets (the first value in one subset and the other four in the other subset).

The process then determines (at 2030) the set $S$ of subsets that satisfy particular constraints for the problem being solved. In some embodiments, the purpose of the constraints is to determine subsets that are relatively denser than the other subsets. Some embodiments use two density constraints: a minimum cluster size (i.e., the minimum number of values in the subset) and maximum cluster spread (i.e., the largest allowed difference between the largest and smallest values in the subset). In the case of using density clustering for determining alignment guides, some embodiments use a minimum cluster size that is a fraction of the total lines in the page or
zone being evaluated, while other embodiments use a constant. Some embodiments use a maximum spread that is a fraction of the median font size of the first (for left-alignment) or last (for right-alignment) characters of words.

[00207] Once the set $S$ of subsets that satisfy the constraints are determined, the process determines (at 2035) whether $S$ is empty. When $S$ is empty, the process proceeds to 2055 which is described below. When $S$ includes at least one subset, the process evaluates (at 2040) an optimization function for $S$. Some embodiments use an optimization function that looks for the set $S$ that has the largest subset that meets the constraints. Other embodiments use an optimization function tries to maximize the sum of the squares of a particular value (e.g., the size of the subset minus the minimum cluster size) over all of the subsets that meet the constraints. Yet other embodiments use one of the above-mentioned optimization functions, and then use the other in case of a tie. Other optimization functions are used by other embodiments.

[00208] Next, the process determines (at 2045) whether the set $S$ is the most optimal so far, based on the optimization function. When $S$ is not the most optimal, the process proceeds to 2055 which is described below. Otherwise, when $S$ is the most optimal, the process stores (at 2050) $S$ as the best set of clusters yet found. The first pass through (in which $d$ is the largest difference) will always be the most optimal at that point, if $S$ is not empty. On subsequent passes, the current $S$ will be compared to the stored set of clusters.

[00209] The process then determines (at 2055) whether there are any unevaluated differences. Some embodiments test each possible partition to find the most optimal set of clusters. Some such embodiments use the efficiency techniques described below in Section X to enable faster and more efficient processing. When the process determines that there are unevaluated differences, the process proceeds to 2020 which was described above.

[00210] Otherwise, once all the differences have been evaluated, the process outputs (at 2060) the currently stored optimal set (or empty set if no clusters satisfying the constraints were found) as the final set of clusters. In the case of determining alignment guides, the final set of clusters would be groups of words with very close x-coordinates. The process then ends. One of ordinary skill will recognize that in addition to the density constraints and optimal measure, process 2000 imposes a consistency constraint on the clusters: namely, that intra-cluster differences between successive values in a cluster will never equal or exceed inter-cluster
differences, because the data is always partitioned at all differences that are equal to or greater than a specified gap minimum.

**B. Determining Alignment Guides**

As mentioned above, some embodiments determine right- and left-alignment guides by searching for associated sets of glyphs (e.g., words, text lines) that start or end at the same or nearly the same x-coordinate on a page and determining whether sufficient evidence exists that the x-coordinate is actually an alignment point. Some embodiments use similar but not identical processes to find left-alignment guides and right-alignment guides.

Figure 21 conceptually illustrates a process 2100 of some embodiments for determining left-alignment guides. Portions of process 2100 will be described in conjunction with Figures 22-24. Figures 22-24 illustrate the process of identifying a left-alignment guide on a page 2200. As shown in Figure 21, process 2100 sets (at 2105) the input data for density clustering as the x-coordinates of the left edge of words in a region of a document. The region is a page or a zone of a page in some embodiments. In some embodiments, the left edge of a particular word is the x-coordinate of the anchor of the first glyph in the particular word, adjusted to the left alignment position expected for the glyph.

The process then determines (at 2110) desired cluster properties. In some embodiments, the cluster properties are the constraints for density clustering described above. Some embodiments use two density constraints: a minimum cluster size (i.e., the minimum number of values in the subset) and maximum cluster spread (i.e., the largest allowed difference between the largest and smallest values in the subset). In the case of using density clustering for determining alignment guides, some embodiments use a minimum cluster size that is a fraction of the total lines in the page or zone being evaluated, while other embodiments use a constant. Some embodiments use a maximum spread that is a fraction of the median font size of the first (for left-alignment) or last (for right-alignment) characters of words. One example of constraints is that the minimum cluster size is 5% of the total number of text lines in the region, and the maximum spread is 10% of the median font size.

Next, the process applies (at 2115) density clustering to the input data using the determined cluster properties to determine clusters of x-coordinate values that may be alignment
guides. Some embodiments use process 2000 as described above.

[00215] Process 2100 then determines (at 2117) whether there are any unevauated clusters. When there are no clusters, or all clusters are evaluated, the process ends. Otherwise, the process selects (at 2120) a cluster (i.e., one of the clusters output from the cluster analysis). The process then sets (at 2125) a left-alignment guide as a rectangle with the minimum and maximum x-coordinates as the smallest and largest values in the cluster and the minimum and maximum y-coordinates as the top and bottom of the page. In some cases, the minimum and maximum x-coordinate will be the same, as all the x-coordinates in the cluster will have the same value. In other cases, small aberrations or words that accidentally make it into the cluster will give the rectangle a non-zero width.

[00216] Figure 22 illustrates a page 2200 with a potential left-alignment guide 2205 in some embodiments. The minimum x-coordinate of the rectangle 2205 is set by the left edge of the right column 2215, while the maximum x-coordinate is set by the word "tate" 2210 in the middle of the page, because the start of word 2210 is close enough to the start of the words forming the left edge of the right column that it is grouped in with those words by the density clustering process.

[00217] Process 2100 then removes (at 2130) the rectangle at y-coordinates that do not satisfy constraints based on an analysis of words that start in the rectangle and words that cross the rectangle. The process then proceeds to 2117, described above. Some embodiments remove a portion of the rectangle anywhere that a word starts left of the rectangle and crosses into the rectangle. The rectangle is also removed at any y-coordinate that is between two crossing words that do not have a sufficient number of border words between them. A border word is a word that starts in or at one of the edges of the rectangle. Some embodiments use a requirement that there be at least five border words between crossing words, and at least one of those five border words must be the leftmost on its text line or separated from the previous word on its text line by more than a normal word gap. Some embodiments use processes described in United States Publication No. 2007/0250497, entitled “Semantic Reconstruction”, by Mansfield, et al., which is incorporated herein by reference, to determine word gaps and larger gaps. Some embodiments use different requirements (e.g., fewer or greater than five border words between crossing words) to perform operation 2130.
[00218] Figure 23 illustrates the page 2200 and rectangle 2205 with the crossing words for rectangle 2205 circled. The crossing words include words 2340 ("reprehendcrit") and 2315 fdolorc"). among others. There are two border words 2210 ("tate") and 2325 ("cssc") between crossing words 2340 and 2335; however, when the requirement for border words in between crossing words is three or larger, the rectangle would be removed through this section as well. Some embodiments remove only from the greatest ascent to the greatest descent of crossing words and non-qualifying areas in between crossing words. Other embodiments also remove areas that are likely beyond the alignment guides, such as the area from the crossing word 2330 ("autcir") to the border word 2335 ("reprchenderit") above it.

[00219] Figure 24 illustrates left-alignment guides 2405 and 2410 for page 2200. Because of the call-out region in the center of the page, the left-alignment guides at that particular x-coordinate do not run the length of the entire page 2200.

[00220] As mentioned above, some embodiments use a process similar to process 2100 for determining right-alignment guides. Figure 25 conceptually illustrates a process 2500 of some embodiments for determining right-alignment guides. As shown, the process sets (at 2505) the input data for density clustering as the x-coordinates of the right edge of words in a region of a document. The region is a page or a zone of a page in some embodiments. In some embodiments, the right edge of a particular word is the x-coordinate of the anchor of the last glyph in the particular word plus the x-coordinate of the advance vector for the last glyph in the word, adjusted to the right alignment position expected for the glyph.

[00221] The process then determines (at 2510) desired cluster properties. In some embodiments, the cluster properties are the constraints for density clustering described above. Some embodiments use two density constraints: a minimum cluster size (i.e., the minimum number of values in the subset) and maximum cluster spread (i.e., the largest allowed difference between the largest and smallest values in the subset). In the case of using density clustering for determining alignment guides, some embodiments use a minimum cluster size that is a fraction of the total lines in the page or zone being evaluated, while other embodiments use a constant. Some embodiments use a maximum spread that is a fraction of the median font size of the first (for left-alignment) or last (for right-alignment) characters of words. One example of constraints are that the minimum cluster size is 5% of the total number of text lines in the region, and the
maximum spread is 10% of the median font size.

[00222] Next, the process applies (at 2515) density clustering to the input data using the determined cluster properties to determine clusters of x-coordinate values that may be alignment guides. Some embodiments use process 2000 as described above.

[00223] The process then determines (at 2517) whether there are any unprocessed clusters. When there are no clusters, or all clusters have been processed, the process ends. Otherwise, the process selects (at 2520) a cluster (i.e., one of the clusters output from the cluster analysis). The process then sets (at 2525) a right-alignment guide as a rectangle with the minimum and maximum x-coordinates as the smallest and largest values in the cluster and the minimum and maximum y-coordinates as the top and bottom of the page. In some cases, the minimum and maximum x-coordinate will be the same, as all the x-coordinates in the cluster will have the same value. In other cases, small aberrations or words that accidentally make it into the cluster will give the rectangle a non-zero width.

[00224] The process then removes (at 2530) the rectangle at y-coordinates that do not satisfy constraints based on an analysis of words that end in the rectangle and words that cross the rectangle. The process then proceeds to 2517, described above. Some embodiments remove a portion of the rectangle anywhere that a word crosses or starts in the rectangle and ends right of the rectangle. The rectangle is also removed at any y-coordinate that is between two crossing words that do not have a sufficient number of border words between them. A border word is a word that ends in or at one of the edges of the rectangle. Some embodiments use a requirement that there be at least five border words between crossing words, and at least one of those five border words must be the rightmost on its text line or separated from the next word on its text line by more than a normal word gap. Some embodiments use processes described in the above-mentioned United States Publication No. 2007/0250497 to determine word gaps and larger gaps.

Some embodiments use different requirements (e.g., fewer or greater than five border words between crossing words) to perform operation 2530.

C. Determining Gutters

[00225] After determining the guides, some embodiments then determine gutters of the region (e.g., zone, page, etc.). Some embodiments use information from the guide determination
process (e.g., processes 2100 and 2500) to determine the groupings of unfilled white space between associated glyphs (c.g., gutters) of the region. Some embodiments also use other alignment points in addition to guides for determining gutters in a region.

[00226] **Figure 26** conceptually illustrates a process 2600 of some embodiments for determining gutters for a region. Portions of process 2600 will be described in conjunction with **Figures 27-29.** **Figures 27-29** illustrate the process of identifying a gutter on a page 2700.

[00227] As shown in **Figure 26,** the process receives (at 2605) alignment information. In some embodiments, this information is the guides determined by processes 2100 and 2500. Some embodiments include other alignment points as well as guides. For instance, in some embodiments, the end of text lines in left-aligned (not justified) text are treated as right-alignment points. This enables gutters to be identified in column gaps even if no guide is found at the right edge of the first column. Similarly, the left edge of right-aligned text, or both edges of centered text, are considered alignment points in some embodiments.

[00228] Process 2600 then determines (at 2607) whether there are any unprocessed right-alignment points. When there are no right alignment points, or all have been processed, the process ends. Otherwise, the process selects (at 2610) a right-alignment point. In some embodiments, the process identifies the leftmost right-alignment point first, while in other embodiments it picks a random right-alignment point.

[00229] The process then determines (at 2615) whether a left-alignment point exists between the selected right-alignment point and the right edge of the region. When there are no left-alignment points, the process proceeds to 2607, which was described above. Otherwise, when there is at least one left-alignment point between the right-alignment point and the region edge, the process identifies (at 2620) the next left-alignment point moving right across the region from the selected right-alignment point. It is the area between these two points that the process tests to determine if there is a gutter.

[00230] Once the right- and left-alignment points are identified, the process sets (at 2625) a gutter as a rectangle with the right-alignment point as the minimum x-coordinate and the left-alignment point as the maximum x-coordinate. The minimum and maximum y-coordinates of the rectangle are the top and bottom of the page. **Figure 27** illustrates the page 2700 and a rectangle
2705 that is to be tested as a possible gutter. The minimum x-coordinate is the right-alignment point at the right edge of the first column, and the maximum x-coordinate is the left-alignment point at the left edge of the second column.

[00231] Next, the process removes (at 2630) the gutter at y-coordinates that do not satisfy constraints based on an analysis of words that cross into the rectangle and border the rectangle. Some embodiments remove a portion of the rectangle anywhere that a word crosses into or starts in the rectangle. The rectangle is also removed at any y-coordinate that is between two crossing words that do not have a sufficient number of border words between them. A border word for a gutter is a word that ends at the left edge of the rectangle or starts at the right edge of the rectangle. Some embodiments use a requirement that there be at least five border words between crossing words, and at least one of those five border words must be either the leftmost on its text line or separated from the previous word on its text line by more than a normal word gap or the rightmost on its text line or separated from the next word on its text line by more than a normal word gap. Some embodiments use processes described in the above mentioned United States Publication No. 2007/0250497, to determine word gaps and larger gaps. Some embodiments use different requirements (e.g., fewer or greater than five border words between crossing words) to perform operation 2630. The process then proceeds to 2607, which was described above.

[00232] Figure 28 illustrates the page 2700 and rectangle 2705 with the crossing words for rectangle 2705 circled. The crossing words include words 2810 ("cillum") and 2815 ("nulla"), among others. There is a border word 2820 ("eu") between crossing words 2810 and 2815; however, if the requirement for border words in between crossing words is two or larger, then the rectangle would be removed through this section as well. Some embodiments remove only from the greatest ascent to the greatest descent of crossing words and non-qualifying areas in between crossing words. Other embodiments also remove areas that are likely beyond the gutters.

[00233] Figure 29 illustrates gutters 2905 and 2910 for page 2700. Because of the call-out region in the center of the page, the gutter between the two main columns does not run the entire length of the page.

[00234] Some embodiments use the guides and gutters throughout the semantic reconstruction process. For example, gutters arc used to snlit text lines and identity columns.
processes that are described below in Section IV.

D. Software Architecture

In some embodiments, the guide and gutter analysis processes described above are implemented as software running on a particular machine, such as a computer, a media player, a cell phone (e.g., an iPhone®), or other handheld or resource-limited devices (or stored in a computer readable medium). Figure 30 conceptually illustrates the software architecture of a guide and gutter analysis application 3000 of some embodiments for identifying guides and gutters in a document. In some embodiments, the application is a stand-alone application or is integrated into another application (e.g., a document reconstruction application), while in other embodiments the application might be implemented within an operating system.

Guide and gutter analysis application 3000 includes a guide identification module 3005, a density clustering module 3010, and a gutter identification module 3015, as well as guide and gutter information storage 3020.

Figure 30 also illustrates document content 3025. Guide identification module 3005 receives information from the document content 3025. The guide identification module 3005 analyzes the document content to identify alignment guides in the document. The identified guides are passed to gutter identification module 3015 as well as to guide and gutter information storage 3020 and to the document content 3025. In some embodiments, guide identification module 3005 performs some or all of processes 2100 and 2500.

The guide identification module 3005 also passes information to, and receives information from, the density clustering module 3010. Density clustering module 3010 receives input data from the guide identification module 3005 and/or the guide and gutter information storage 3025 and performs density clustering on the input data in order to determine potential guides. In some embodiments, density clustering module 3010 performs some or all of process 2000.

The gutter identification module 3015 receives information from the guide identification module 3005 and the document content 3025. The gutter identification module analyzes the received information to identify gutters in the document. The identified gutters are passed to the guide and gutter information storage 3020 and to the document content 3025. In
some embodiments, gutter identification module 3015 performs some or all of process 2600.

[00240] In some embodiments, the results of the processes performed by the above-described modules or other modules are stored in an electronic storage (e.g., as part of a document object model). The document object model can then be used for displaying the document on an electronic display device (e.g., a handheld device, computer screen, etc.) such that a user can review and/or interact with the document (e.g., via touchscreen, cursor control device, etc.).

IV. DETERMINING THE LAYOUT AND FLOW

[00241] Documents generally have an implicit structure and flow of content. Specifically, in some cases, ordered sequences of characters (and inline graphics) make up words, ordered sequences of words make up text lines (or span text lines with a hyphen), ordered sequences of text lines make up paragraphs, ordered sequences of paragraphs make up columns (or span columns), ordered sequences of columns make up layouts, and ordered sequences of layouts make up sections of a document. When this structure is not provided in the file format of an electronic document, the structure has previously been inaccessible to software. While merely viewing a document does not necessarily require document structure, applications for editing, importing, searching, styling, or otherwise repurposing a document do require knowledge of the document structure and flow in order to function properly.

[00242] Some embodiments of the invention provide methods for determining the layout and flow of a document or a region of a document. This includes determining the semantic hierarchy (e.g., the words, lines, and paragraphs of a document), as well as layout properties such as the columns and how the columns fit together for intended reading of the document. In some embodiments, the goal of the processes is to identify the order in which a human would read a document from start to finish.

[00243] Figure 31 conceptually illustrates a process 3100 of some embodiments for determining the layout and flow of a document. Process 3100 will be described in conjunction with Figure 32. Figure 32 illustrates a sequence of various layout and flow information being determined for a page 3200 of a document with two columns of text. In Figure 32, one will recognize that the content of page 3200 is not important, but rather that the lines, paragraphs, etc.
arc of import. As shown in Figure 31, process 3100 receives (at 3105) a portion of a document. In some embodiments, the portion is the entire document, or a section, page, or zone.

[00244] The process then identifies (at 3110) lines of text in the received document. This includes identifying characters that share a common baseline and merging preliminary lines together when necessary (e.g., subscripts and superscripts). Figure 32 illustrates the identification of lines 3205 and 3210. The line identification process of some embodiments is described in further detail below in subsection A.

[00245] Next, the process identifies (at 3115) words in the text. Some embodiments use difference clustering, as described in above mentioned United States Publication No. 2007/0250497 to identify words in the text. Figure 32 illustrates the identification of words on page 3200, including the word 3215 ("Lorcm") from line 3205 and the word 3220 ("amct") from line 3210. The word identification process is also described in further detail below in subsection B.

[00246] The process then splits (at 3120) the lines of text where the text is discontinuous. Figure 32 illustrates that line 3205 is split into lines 3225 and 3230, and line 3210 is split into lines 3235 and 3240. The line splitting process of some embodiments is described in further detail below in subsection C.

[00247] After splitting the lines, the process places (at 3125) the text lines into paragraphs. Figure 32 illustrates paragraphs 3245 and 3250 identified on page 3200. The paragraph identification process is described in further detail below in subsection D.

[00248] Lastly, the process places (at 3130) the paragraphs into columns and layouts. Figure 32 illustrates columns 3255 and 3260 identified on page 3200. The column and layout identification process is described in further detail below in subsection E.

[00249] Some embodiments do not perform all of the operations of process 3100 at once. Instead, some perform other document reconstruction processes in between operations of process 3100. For example, some embodiments determine lines of text and the words in the text, but then identify guides and gutters prior to splitting the lines of text.

A. Initial Line Identification
[00250] As mentioned above, in some embodiments lines of text have to be identified. Because every character in a particular line of text will not necessarily always share a common baseline, some embodiments attempt to merge lines together based on evidence that the characters in the two lines are intended to be read as part of the same line of text (e.g., superscripts and subscripts).

[00251] **Figure 33** conceptually illustrates a process 3300 of some embodiments for identifying and merging lines of text. Process 3300 will be described in conjunction with **Figures 34 and 35. Figure 34** illustrates a page 3400 with six groups 3405-3430 of overlapping text lines, and **Figure 35** illustrates the merging of those groups of text lines according to some embodiments of the invention.

[00252] As shown in **Figure 33**, the process receives (at 3305) a portion of a document. In some embodiments, the portion is a page of a document, or a zone of a page, etc. The process then determines (at 3307) whether there are any characters in the document portion. When there are none, the process ends. Otherwise, the process associates (at 3310) as preliminary text lines characters that share a common baseline. Characters share a common baseline in some embodiments when they have the same y-coordinate anchor point. In general, associating characters that share a common baseline will group together lines of standard text. Some embodiments use a small threshold such that the y-coordinate anchor points in a preliminary text line need not be exactly equal, but must be within the small threshold of each other.

[00253] Next, the process identifies (at 3315) groups of text lines that vertically overlap. Two lines vertically overlap in some embodiments when the bounding rectangle of the first line overlaps in y-coordinate values with the bounding rectangle of the second line. **Figure 35** illustrates the page 3400 with six groups of vertically overlapping text lines: lines 3505 and 3506, lines 3510 and 3511, lines 3515 and 3516, lines 3520, 3521, and 3522, lines 3525 and 3526, and lines 3530 and 3531. Line 3520 is associated in a group with line 3522 because both overlap with line 3521, even though they do not overlap each other. Even though there is no horizontal overlap, because lines 3530 and 3531 vertically overlap, they are initially grouped together in some embodiments.

[00254] The process then selects (at 3320) an unevauated group and partitions (at 3325) the group into sections with no horizontal overlap between text lines. **HifFornt <orfinn>** Two
text lines horizontally overlap in some embodiments when the x-coordinates of the bounding box of the first text line overlap with the x-coordinates of the bounding box of the second text line. For instance, lines 3530 and 3531 are partitioned at this point because they do not horizontally overlap and thus would not be likely to be considered the same line. Some embodiments expand the measure of horizontal overlap a small distance (e.g., one half of a space character) at the beginning and end of the text lines, so that offset characters (e.g., subscripts and superscripts) at the beginning or end of a line are merged. For example, there is no horizontal overlap between lines 3510 and 3511, but they are not partitioned because the end of line 3510 is close enough to the beginning of line 3511.

After partitioning the selected group, the process selects (at 3330) an unevaluated section from the group and sorts (at 3335) the lines in the section from top to bottom. Thus, if the selected section with lines 3520-3522 is selected, the lines would be sorted with line 3520 first, line 3521 second, and line 3522 third. Various embodiments sort the lines by ascent, descent, baseline, or other measure of the vertical position of a line.

The process then selects (at 3340) the top-most unevaluated line in the section. Next, the process selects (at 3345) the first (reading from the left for left-to-right languages) unevaluated character in the selected line. The process determines (at 3350) whether the selected character can be merged into the next line. Some embodiments allow a character to be merged into the next line when the selected character does not horizontally overlap significantly with any character in the next line. Some embodiments allow some small amount of horizontal overlap between characters. For left-to-right languages, some embodiments allow less overlap on the left of the character to be merged down than on the right of the character to be merged down, in order to account for common spacing adjustments for offset characters.

Furthermore, some embodiments allow any amount of overlap when the original insertion order of the overlapping characters is adjacent. The insertion order, in some embodiments, is the order in which the characters are drawn on the page. Often (though not always), characters are drawn in the order they are meant to be read, so when two vertically and horizontally overlapping characters are adjacent in the insertion order, it is likely they are intended to be read together.

When the process determines that the selected character can be merged into the
next line, the process merges (at 3355) the selected character into the next line. The process then proceeds to 3365 which is described below. Otherwise, when the selected character cannot be merged, the process keeps (at 3360) the selected character in the selected line.

[00259] Next, the process determines (at 3365) whether the selected line includes more characters. When there are more characters in the currently selected line, the process proceeds to 3345 to select the next unevaluated character in the line. Otherwise, when all characters in the line have been evaluated, the process determines (at 3370) whether the current section includes more lines. When there are more lines in the currently selected section, the process proceeds to 3340 to select the next unevaluated line.

[00260] Otherwise, when all lines in the section have been evaluated, the process determines (at 3375) whether the selected group includes more sections. When there are more sections in the currently selected group, the process proceeds to 3330 to select another section and merge lines in that section. Otherwise, when all the sections in the group have been evaluated, the process determines (at 3380) whether there are any more groups to evaluate in the document portion. When there are more groups, the process proceeds to 3320 to select another group. Otherwise, when all groups have been evaluated, then line-merging is finished for the document portion and the process ends.

[00261] Figure 35 illustrates the result of line merging for page 3500 in some embodiments. Line 3506 can merge down into line 3505, such that line 3505 now includes the superscript "m" from line 3506, while line 3506 is empty and is therefore removed. Although there is no horizontal overlap between lines 3510 and 3511, the end of line 3510 is close enough to the start of line 3511 that they are not partitioned, and all of line 3510 can be merged down into 3511. Both characters in line 3516 are merged down into line 3515.

[00262] Lines 3520-3522 cannot be fully merged. The character "b" in line 3520 is initially merged down into line 3521. Then, the character "A" in line 3521 is merged down into line 3522 as it does not overlap with the character "c". However, character "b" is not merged down into line 3522 because it completely overlaps with character "c". Thus, line 3521 only includes "b", line 3522 includes "A" and "c", and line 3520 is empty. As described above, some embodiments will merge "b" into line 3522 if "b" and "c" are adjacent in the insertion order.
Similarly, lines 3525 and 3526 are not merged. All of the characters in line 3526 significantly overlap one or more characters in line 3525, and therefore are not merged down into line 3525. It is unlikely that the "T" in line 3526 would be between the "h" and "n" of line 3525 in the insertion order for page 3500. Lastly, lines 3530 and 3531 are not merged because there is no horizontal overlap between the lines and thus they are partitioned at operation 3325.

After the lines are identified and merged, words are identified in some embodiments. Some embodiments use difference clustering, as described in United States Publication No. 2007/0250497 to identify words based on spacing between letters within a word and between words. In some embodiments, the difference clustering also provides information about segment gaps, column gaps, etc. Some embodiments use the memory and processing efficiency techniques described below in Section X to perform difference clustering.

B. Identifying Words and Gaps Using Difference Clustering

Figure 36 conceptually illustrates a process 3600 of some embodiments for performing difference cluster analysis. Many forms of cluster analysis require foreknowledge of the number of groups/clusters since there may exist multiple levels/hierarchies of clustering. For example, when using cluster analysis to group celestial objects, a specification of the number of clusters determines whether the cluster analysis will group objects on the level of stars, solar systems, galaxies, or superclusters. However when using cluster analysis to discover the structural relationships between elements of content, e.g., the number of groups are not known in many cases. For example, in the case of a page of text, it cannot be assumed the glyphs make up words, words combine to form lines, and groups of lines form paragraphs, because the document may have two or more columns of text such that a given initial line of text may include parts of two or more paragraphs.

In some embodiments, cluster analysis is a set of techniques that can be applied to a collection of data points to group points into clusters that are closer to each other than to the points of another cluster. In some embodiments, cluster analysis is applied to data points that represent the horizontal and vertical gaps between objects such as glyphs, words, and text lines. For example, some embodiments use k-mcans cluster analysis, which will now be described. Starting with a collection of numbers (pi, . . . , px) representing spatial gaps, and a known value for k (the number of clusters), the technique is used to partition the numbers into k clusters Ci. . .
, $C_k$ defined by inequalities of the form $C_k = \{ p_i \mid a_i \leq p_i < a_{i+1} \}$ where $a_1, \ldots, a_{k-1}$ is an increasing sequence. Before applying the k-mcans technique, the differences $p_{i-1} - p_i$ are sorted by size and the k-l largest differences are taken to be the partition points. For example, if $P_M - P_i$ is one of the k-l largest differences, then $p_{i-1}$ is in a different cluster from $p_i$, and $p_{i+1}$ is one of the successive values $a_j$. k-mcans cluster analysis is then applied to repeatedly refine the clusters. The k-means technique involves taking the mean of the numbers in each cluster, then redistributing the $p_i$ into clusters by associating them with the closest calculated mean. This is performed repeatedly until it causes no change in the clusters or their means.

In some embodiments, a technique disclosed and referred to herein as "difference clustering" is used to determine the number of levels of structural relationships that exist between content elements comprising a given source content and/or one or more hierarchical relationships between such levels, as well as one or more characteristics that can be used to determine whether a content element is related to another content in each of the determined levels. In some embodiments, difference clustering utilizes the k-mcans technique together with other techniques. In the example shown in Figure 36, differences between positions of content elements (spacing) are analyzed using difference clustering analysis. In some embodiments, by analyzing the spacing between content elements, the content elements can be grouped at least in part using the grouping data of the spacing. In some embodiments, each directional component of spacing is analyzed separately. For instance, difference clustering analysis on the horizontal component is used to distinguish between character spacing, word spacing, and column spacing. Difference clustering analysis on the vertical component can be used to distinguish line spacing, paragraph spacing, and text box spacing in some embodiments. Process 3600 conceptually illustrates difference clustering analysis for a single directional component. The process may be used again to analyze one or more additional directional components. In some embodiments, the results of performing difference cluster analysis along one or more dimensions are combined together to determine the structural relationships between content elements at one or more levels.

As shown in Figure 36, process 3600 receives (at 3605) a portion of a document. The process then identifies (at 3610) the locations of elements in the document. In some embodiments, the elements include characters, glyphs, images, lines, drawings, boxes, cells, margins, and/or various other content elements. In some embodiments, locations of the elements
include determining and/or assigning one or more location coordinate components to the elements. In some embodiments, the locations of the elements are organized in an order. For example when analyzing the horizontal spacing of characters, the characters are organized in increasing horizontal coordinate order for each line of characters. In some embodiments, the location coordinate values of the elements are desired to be associated with the spacing between the elements, and the location values are compensated for the width/length of the element. For example, when determining a compensated horizontal coordinate (x-coordinate) value for an element in the n-th position of an organized order of elements, the following formula is used:

\[ x_n' = x_n - \sum_{i=1}^{n-1} W_i \]

where \( x_n' \) is the compensated location coordinate value, \( x_n \) is the original location coordinate value, and \( W_i \) is width of an element in the i-th position. In some embodiments, the width of an element is based on the character it represents, the font size, the styling of the character, etc. Some embodiments determine a compensated location coordinate value by using known anchor coordinates for each character, and adjusting those coordinates for each particular character by the width of the particular character.

[00269] Next, the process determines (at 3615) the first-order differences between locations of adjacent elements. In some embodiments, an element is adjacent to another element when the two elements with at least one same location coordinate component value arc ordered next to each other in at least one other location coordinate component value. For instance, two glyphs are adjacent to each other if both of the glyphs belong to the same text line and no other glyph exists between them. In some embodiments, two elements have at least one same location coordinate component when the difference between corresponding location coordinate component values of the elements is below a limit value or within a range value. In various embodiments, an element is adjacent to another element when the two elements arc next to each other in an order and/or organization associated with the identified locations of the elements. In some embodiments, the first order difference between the locations is the difference between the width/length compensated location coordinate values. For instance, when determining the difference between compensated horizontal coordinate (x-coordinate) values for the adjacent
elements in the nth and n+1 position of an organized order of compensated horizontal coordinates, in some embodiments the following formula is used.

$$\Delta X_n = X_{n+1}^i - X_n^i$$

In some embodiments, the first order difference is associated with the gap spacing between glyphs in the content.

[00270] Next, process 3600 sorts (at 3620) the first order differences. In some embodiments, organizing the first order difference includes ordering the first order differences in an increasing order. In some embodiments, organizing the first order differences includes assigning a weight value to one or more of the first order differences and organizing the first order differences at least in part by using the weight value(s). For instance, in some embodiments, actual glyph spacing is divided by expected glyph spacing for each specific pair of glyphs given the font that is used and its font metrics including size, default letter spacing, and a table of kerning values stored with the font file. This ratio of actual to expected spacing is ordered by increasing value, and the values of this ratio are used in place of the first order differences throughout the remainder of the difference clustering method.

[00271] The process then determines (at 3625) second order differences between the sorted first order differences. For instance, when determining the second order difference between first order differences in an i-th and i+1 position of an organized order of first order differences, the following formula is used:

$$\Delta^2 X_i = \Delta X_{i+1}^s - \Delta X_i^s$$

where $\Delta^2 X_i$ is the i-th second order difference, $\Delta X_{i+1}$ is the first order difference in the i-th position of the sorted first order differences, and $\Delta X_{i+1}$ is the first order difference in the i+1 position of the same sorted first order differences. In some embodiments, the second order differences are associated with differences between the spacing of glyphs.

[00272] Next, process 3600 determines (at 3630) the number of cluster levels by analyzing the second order differences. In some embodiments, analyzing the second order differences includes organizing the determined second order differences. In some embodiments, organizing

56
the second order difference includes ordering the second order differences in an increasing order and/or plotting the second order differences in an order of increasing second order difference values. In some embodiments, organizing the second order difference includes assigning a weight value to one or more of the second order difference. In some embodiments, organizing the second order difference includes grouping the second order differences into one or more groups. In some embodiments, the second order differences are each categorized as either an inter-group difference or an intra-group difference.

[00273] Intra-group differences are associated with relatively smaller second order difference values and can represent second order differences of first order differences within the same clustering group. An example of an intra-group difference is the relatively small variation one would expect to find in the character-width compensated spacing between letters in the same word. Inter-group differences are associated with relatively larger difference values and can represent second order differences of first order differences between different clustering groups. An example of an inter-group difference is the relatively large difference between the space between two words, on the one hand, and the space between two letters in the same word, on the other.

[00274] In some embodiments, the categorization of second-order differences into intra-group and inter-group values is achieved by applying 2-means cluster analysis to the ordered second-order difference values: specifically, taking \( p_i, \ldots, p_N \) to be \( \{\Delta^2 X_1, \ldots, \Delta^2 X_N\} \) in increasing order. Similarly, any other technique of cluster analysis that is sufficient to distinguish two clusters of data values can be applied to the ordered second-order difference values. The intra-group differences are then in the first cluster \( C_1 = \{p_i \mid a_1 \leq p_i < a_2\} \), and the inter-group differences are in the second cluster \( C_2 = \{p_i \mid a_2 \leq p_i < a_i\} \), where \( a_i < a_2 < a \). In some embodiments, the number of levels into which content elements are determined to be organized, based on their spatial relationships analyzed as described above, is one more than the number of inter-group differences found through difference cluster analysis. For instance, when two inter-group differences exist, the number of structural levels is three. Taking a simple example, consider characters that form words comprising a single line of text. The first order differences in the spacing between characters in the x-x-direction would yield a second order difference between character spacing and word spacing (one inter-group difference), indicating two levels
of structure (words and lines). When the text had been in two columns, a further second-order
difference (between word spacing and column spacing) would have been detected, for a total of
two inter-group differences, indicating three structural levels in the x-direction (words, lines, and
columns). Repeating the analysis in the y-direction and combining results would, when
applicable to the particular content, identify in some embodiments any further structural levels
(e.g., paragraphs, etc.) that are manifested in the spacing between characters and groups of
characters.

[00275] The process then determines (at 3635) characteristics of each cluster level. The
process then ends. In some embodiments, determining the characteristics includes determining
which first order difference (and/or what range of first order differences) is associated with
which cluster level. In some embodiments, determining the characteristic includes computing a
statistical value associated with the first order differences associated with a cluster level. For
example, by determining the average, minimum, maximum of the portion of first order
differences associated with a cluster level, the average, minimum, and maximum spacing
between glyphs in the content can be determined.

[00276] Let L be the number of levels of clustering. In some embodiments, L is computed
by counting the number of points in the second cluster of second-order differences and adding 1.
Next, the groups of first-order differences corresponding to each level can be identified, and the
clusters of compensated $X'_n$ values can be identified at each level, for example, in one of the
following two ways.

[00277] One possibility is to perform L-means cluster analysis on the first-order
differences. The resulting L clusters are the groups of first-order differences corresponding to
each level. Next the number $K_m$ of clusters of $X'_n$ at level m are computed by adding the number
of points in the (m+1)th, (m+2)th, . . . , and Lth clusters of first-order differences plus 1. Finally,
perform $K_n$-means analysis on the compensated $X'_n$ values to produce the $K_n$ clusters at level m.

[00278] A second possibility is, when originally computing each first-order difference
$\Delta X_n = X_{n'} - X'_n$, to store its value together with the index n that can be used to identify either
one of the pair of successive X values that were subtracted to produce that difference. Store the
value and the index reference in a single "first-order difference" data structure. Similarly, when
originally computing each second-order difference, store its value together with an index
reference that can be used to identify either one of the pair of successive "first-order difference" data whose values were subtracted to produce that difference. Now, for each second-order difference that is in the second cluster (i.e. for each inter-group difference), use its index reference to identify a partition point in the first-order differences. This means that the index identifies a pair of first-order difference values that are partitioned to be in separate clusters. Partitioning in this way produces L clusters of first-order differences corresponding to the L levels of clustering in the original data. Now, the clusters of $X_n$ values at level n are identified as follows: for each first-order difference data in the $(m+1)$th, $(m+2)$th, ..., and $L$th cluster of first-order differences, use its index reference as a partition point in the $X_n$ values.

Figure 37 illustrates an example of difference clustering. In some embodiments, the example of Figure 37 is associated with process 3600 of Figure 36. Groups of first order difference values 3705, 3710, and 3715 are plotted in order from lowest value to highest value on a line associated with first order difference values. Each point is associated with a difference value, e.g., the distance from a text character or other glyph to an adjacent one, and in Figure 37 the points are not super-imposed on top of each other to illustrate the example clearly.

In the example shown, the data are associated with horizontal spacing between glyphs. By ordering the first order difference values, the example illustrates three groups of first order difference values 3705, 3710, and 3715. First order difference value group 3705 is associated with spacing between glyphs that compose words. First order difference value group 3710 is associated with spacing between words. First order difference value group 3715 is associated with spacing between columns. For each pair of adjacent first order difference values, a second order difference value (i.e., the difference between one first order difference and an adjacent first order difference) is determined and plotted in an increasing order on a line associated with second order difference values. Second order difference value group 3720, 3725, and 3730 each include one or more points associated with the second order difference values. In some embodiments, point 3725 is a member of a group of associated second order difference points comprising a second order difference value group.

In some embodiments, point 3730 is a member of a group of associated second order difference points comprising a second order difference value group. In some embodiments, 3720 is identified as one cluster and 3725 together with 3730 is identified as a second cluster.
Second order difference values between the first order difference values within the same single first order difference value group (intra-group differences) are included in second order difference value group 3720. In a text document, for example, typically the character-width compensated spacing between characters within a word, or in the spacing between different pairs of words, varies only slightly. The second order difference between inter-group adjacent points in group 3705 and 3710 is included in point 3725. The second order difference between inter-group adjacent points in group 3710 and 3715 is included in point 3730. Since there exists two inter-group second order difference values in the example, there are two plus one (three) grouping levels (in this example, words, sentences or parts thereof on a line of text within a column, and columns). By determining the minimum and maximum of the first order difference values in group 3705, minimum and maximum spacing between glyphs that compose words can be determined, and similarly group 3710 and 3715 can be used to determine word spacing and column spacing respectively.

[00282] In some embodiments, the minimum and maximum spacing associated with each grouping level is used to group content elements (e.g., glyphs) accordingly, such as by identifying groups of characters that comprise words, group words into lines of text within a column, etc. By using data determined from cluster analysis, the glyphs are grouped into the determined levels of groupings. It is possible to perform the analysis quickly and automatically with respect to any arbitrary content, in part because it is not necessary to know in advance how many grouping levels there are in the structure of the content or other collection of elements being analyzed. Regardless of the number of grouping levels, the number of levels is determined in just two processing steps. By determining the average of the first order difference values in group 3705, the average spacing between glyphs that compose words can be determined. Similarly, other statistical quantities can be determined for the glyphs that compose words. Similarly, an analysis of the first order difference values in group 3710 and 3715 can be used to determine statistical quantities relevant to word spacing and column spacing.

C. Splitting Lines

[00283] Some embodiments split text lines after word and segment break information is generated. Text lines are split, for example, where the text line spans more than one column, as the text in the two (or more) sections is probably not meant to be read together. Some
embodiments use guide and gutter information derived from processes described above in Section 111 along with information from difference clustering (e.g., segment gaps, etc.) in order to split the text lines.

[00284] Figure 38 conceptually illustrates a process 3800 of some embodiments for splitting lines of text. Portions of process 3800 will be described in conjunction with Figure 39. Figure 39 illustrates a sequence that shows the identification of where lines on a page 3900 should be split. As shown in Figure 38, process 3800 receives (at 3805) text lines, guide and gutter information, and segment break information for a portion of a document. Text line information is the output of process 3300 in some embodiments, and guide and gutter information is the output of processes 2100, 2500, and 2600 in some embodiments. The segment break (or segment gap) information is one of the outputs of difference clustering as described in the above mentioned United States Publication No. 2007/0250497, as well as above, in some embodiments. In some embodiments, the document portion is the entire document, a section, a page, or a zone of a page.

[00285] Next, the process sorts (at 3810) the received text lines based on the y-coordinate of their baselines. Starting at the bottom of the page, the process selects (at 3815) the bottom-most unevaluated text line and identifies (at 3820) potential splits in the selected line. Some embodiments define a potential split as any gap between two words in a line either (1) is a segment gap, as defined by difference clustering, or (2) has a guide or gutter passing through it. Other embodiments only use one or the other, or different definitions, for potential splits.

[00286] The process then determines (at 3822) whether any potential splits were identified. When none were identified, the process proceeds to 3845, described below. Otherwise, the process selects (at 3825) a potential split from the currently selected text line. The process then determines (at 3830) whether the x-interval of the potential split overlaps with the x-interval of any potential split from the previous text line. The first text line evaluated will not have a previous text line, and therefore there will be no overlapping potential splits. When the x-interval of the currently selected potential split does not overlap with the x-interval of a potential split from the previous line, the process proceeds to 3822 which was described above. Otherwise, the process associates (at 3835) the overlapping potential splits. The process then proceeds to 3822 which was described above.
[00287] When there are no more uncorrected potential splits, the process determines (at 3845) whether there are more lines to evaluate. When more lines remain, the process proceeds to 3815 to identify potential splits in the next line and test them for overlap.

[00288] When all lines have been evaluated, then all the potential splits in the document portion have been identified and associated. The process then performs several operations to eliminate false positives (i.e., potential splits that should not actually split a line of text). The process determines (at 3847) whether any groups of potential splits were identified. When none were identified, the process ends. Otherwise, the process selects (at 3850) a group of associated potential splits and defines (at 3855) a rectangular strip passing completely through the potential splits of the selected group. The strip, in some embodiments, has an x-interval that is the intersection of the x-intervals of all the potential splits in the selected group (i.e., the x-interval for a strip two of whose potential splits barely overlap will be very thin).

[00289] Figure 39 illustrates a page 3900 with several lines of text. Most of the lines of text are split between two columns. However, the baselines are the same in either column. The column, each line from the first column would be in the same line as a line from the second column prior to the line-splitting process. Figure 39 also illustrates four rectangular strips 3905, 3910, 3915, and 3920 of associated potential splits.

[00290] After defining the rectangular strip for the selected group, the process determines (at 3860) whether the strip spans fewer than a threshold number of text lines. Strips that span one or only a few text lines are not likely to represent an actual split in reading, but rather may be tabs within a line or other non-breaking gaps. Sometimes segment gaps are found by difference clustering where a gap between words is very large due to justified text. When the strip spans fewer than the threshold number of lines, the process removes (at 3865) the group from the list of potential splits and will not split the text lines at those locations. The process then proceeds to 3890 which is described below. On page 3900, the potential splits making up strips 3910 and 3920 are removed because they do not have enough splits to be a likely column break. More likely, the potential splits are tabs or large word gaps.

[00291] When the strip spans at least the threshold number of lines, the process determines (at 3870) whether the current strip is within a threshold distance of another strip. Some embodiments only look to prior strips that have been tested and not yet removed when
determining whether another strip is within a threshold of the current strip. When the current
strip is within the threshold distance of another strip, the process removes (at 3875) the group
with a vertically shorter strip (in some cases, where the lines are all the same size, this is the strip
that spans fewer text lines). The process then proceeds to 3890 which is described below.

[00292] Strips 3905 and 3915 of page 3900 both qualify as spanning enough text lines to
pass operation 3860. However, in some embodiments the strips are too close to each other to
both be kept. Accordingly, the group of potential splits making up strip 3905 is removed because
3915 is the longer of the two strips. This process prevents list bullets or number from being split
from the items they reference, in some embodiments, as well as other potentially problematic
splits.

[00293] When the current strip is not too close to another strip, the process determines (at
3880) whether the strip includes a threshold number of subsequent potential splits in a row that
are not segment gaps. In some embodiments, it is possible to identify a guide and/or gutter where
word edges accidentally align. This is especially likely if the text is displayed in a monospace
font (e.g., Courier). When the strip includes at least this threshold number of subsequent non-
segment gap potential splits, the process removes (at 3885) the group from the list of potential
splits and will not split the text lines at those locations.

[00294] Next, the process determines (at 3890) whether there are more groups of potential
splits that have not been tested against the various threshold requirements. When more groups
remain, the process proceeds to 3850 to select and evaluate the next group of potential splits.
Otherwise, when all groups have been evaluated, the process splits (at 3895) the text lines using
any of the splits that have not been removed. The process then ends. In the case illustrated for
page 3900, the only splits that would be used are those in the center separating the two columns
of text.

[00295] While process 3800 is illustrated using three specific tests (operations 3860, 3870,
and 3880) to remove groups of potential splits, some embodiments employ only a subset of
these, while other embodiments use other tests that are not shown in order to eliminate potential
splits from consideration.

D. **Paragraph Identification**
In some embodiments, once lines of text have been merged and split, the lines are grouped into paragraphs. Figure 40 conceptually illustrates a process 4000 of some embodiments for grouping text lines into paragraphs. Portions of process 4000 will be described in conjunction with Figure 41. Figure 41 illustrates the identification of paragraphs on a page 4100 of a document. As shown in Figure 40, process 4000 receives (at 4005) text lines for a portion of a document. The text lines have already been merged (e.g., by process 3300) and split (e.g., by process 3800) in some embodiments before process 4000 is performed. In some embodiments, the document portion is an entire document, a section of a document, a page, a zone, etc.

The process determines (at 4007) whether there are any lines in the document portion. When there are none, the process ends. Otherwise, beginning at the top of the received document portion, the process selects (at 4010) the first uncalculated text line in the document portion. The process then determines (at 4015) whether there is more than one text line below the selected line. In some embodiments, the lines must be within a particular vertical distance of each other for the lower line to be considered below the selected line for the purposes of operation 4015. Some embodiments require at least three text lines to make judgments about whether the text lines belong to the same paragraph. In some embodiments, this requirement is imposed because two spacings (i.e., the spacing between the first and second text lines and between the second and third text lines) are necessary in order to make a comparison.

When there are two or more lines below the selected text line, the process proceeds to 4030 which is described below. Otherwise, when fewer than two lines are below the selected text line, the process places (at 4020) the selected line in a paragraph by itself. The process then determines (at 4025) whether there are more lines in the document portion. When there are no more lines (e.g., when there is only one line of text in the document portion), the process ends. Otherwise, when there are more lines, the process proceeds to 4010 and selects the next line of text.

When, at 4015, there are two or more lines of text below the line selected at 4010 (i.e., the first line in the current paragraph), the process identifies (at 4030) the next two lines below the selected text line. The process then determines (at 4035) whether the spacing and alignment is consistent between the three lines. In some embodiments, this determination
involves examining whether the vertical distance from the first to second line is the same as the vertical distance from the second to third line. Some embodiments use the baselines of the text lines to determine the vertical spacing. Alignment differences, in some embodiments, are identified if one of the lines begins indented, or ends left of the other lines, thus signaling a likely beginning or end of a paragraph.

When the spacing and alignment is not consistent, the process applies (at 4040) heuristic rules to determine whether to add either of the identified lines to the paragraph with the selected first line. For instance, in some embodiments, when the first two lines are close together and the third line is further down, the first two lines are placed in one paragraph and the third line is the start of the next paragraph. Similarly, in some embodiments, when the first line is further from the second and third, the first paragraph is a one-line paragraph and the next paragraph starts at the second line. Similar rules are used in some embodiments for alignment differences between the lines. After applying the heuristic rules, the process proceeds to 4010 to select the next unevaluated text line (i.e., the next line that is not yet assigned to a paragraph) and start a new paragraph.

When the spacing and alignment is consistent between the three lines, the process places (at 4045) all three lines in the same paragraph. Some embodiments identify spacing and alignment properties of the paragraph as well. For instance, some embodiments identify paragraphs as left-aligned, right-aligned, justified, centered, etc. Some embodiments leave open multiple possibilities (e.g., a paragraph with an indented first line, all three lines right-aligned or very close, and the lower two lines left-aligned could possibly be any of the three of left-aligned, right-aligned, or justified).

After the initial phase of identifying the start of a new paragraph, process 4000 attempts to add lines to the paragraph. In some embodiments, the line addition is based on the spacing and alignment properties determined from the three lines making up the start of the paragraph. In other embodiments, as lines are added that do not conflict with the spacing and alignment properties for the paragraph, the spacing and alignment properties are refined based on any further evidence.

Next, the process determines (at 4047) whether there are any more lines in the document portion. When there are no more lines (i.e., the document portion has exactly three
lines), the process ends. Otherwise, the process identifies (at 4050) the next text line in the
document portion. The process then determines (at 4055) whether there is a spacing or alignment
mismatch between the current paragraph and the identified next line. When there is a mismatch,
the process ends the paragraph and proceeds to 4010, which was described above. In such a case,
the recently mismatched line will be the line selected at 4010.

[00304] Otherwise, when the spacing and alignment line up, the process adds (at 4060) the
line to the current paragraph. The process then proceeds to 4047, which was described above. In
some embodiments, an alignment mismatch is found when the identified next text line does not fit one of the properties (e.g., justified) of the paragraph. Similarly, if the spacing between the
last line in the paragraph and the next line is increased as compared to that of the paragraph, then
a spacing mismatch is found in some embodiments.

[00305] Some embodiments employ other stopping conditions (e.g., conditions resulting in the identified line not being added to the paragraph). For instance, some embodiments recognize if the first word on the identified line would fit into the white space at the end of the
last line of a left-aligned paragraph. When this is the case, the new line is assumed to be part of
the next paragraph because if it were part of the current paragraph, then the word would be in the
white space at the end of the last line rather than starting a new line. Similarly, some
embodiments recognize an indent as indicating a new paragraph. A third condition of some
embodiments is if the identified line is uniformly styled (e.g., all bold, or of a larger font size)
and different from the styling of any character on the previous line.

[00306] Once process 4000 has completed, all of the paragraphs in the document portion
currently identified, and all lines of text are assigned to a paragraph. Some embodiments then use the
paragraphs to identify columns and layouts.

[00307] Figure 41 illustrates a page 4100 with four paragraphs. Applying process 4000 to
this page (where the page is the document portion) results in the identification of paragraphs
4105, 4110, 4115, and 4120. The process of some embodiments would start by grouping the first
three lines together, then adding the fourth and fifth lines, until the sixth line 4125 had a spacing
and alignment mismatch, leaving paragraph 4105 at five lines. The process would then start with
the sixth line, and notice the spacing and alignment mismatch between the two lines below. As
line six is further from lines seven and eight than they are from each other, line six is the entirev
of paragraph 4.110 and the next paragraph 4.115 starts with line seven. Paragraphs 4.115 and 4.120
are identified similarly.

E. Column and Layout Identification

Some embodiments place paragraphs into columns and layouts after identifying the paragraphs. In some embodiments, a column is a vertically ordered group of paragraphs in which the text reads coherently from the top to the bottom. A layout in some embodiments is a
collection of non-overlapping columns and a linear layout in some embodiments is a horizontally
ordered group of columns in which the text reads coherently from the top of the left-most
column to the bottom of the right-most column. For example, some embodiments classify a
simple page with unsegmented text lines and no headers or footers as a single linear layout with
one column.

Figure 42 conceptually illustrates a process 4200 for identifying columns and
layouts in a portion of a document in some embodiments. Process 4200 will be described in
conjunction with Figures 43-46. Figures 43 and 44 illustrate paragraphs on two different pages
4300 and 4400, and Figures 45 and 46 illustrate the generation of flow graphs for the two pages
4300 and 4400 respectively.

As shown in Figure 42, process 4200 receives (at 4205) information for
paragraphs for the portion of the document. The document portion in some embodiments is an
entire document, a section of a document, a page, a zone, etc. In some embodiments the
paragraph information is determined using process 4000 described above. The process then
determines whether there are any paragraphs to select. When there are none, the process exits.

Otherwise, the process selects (at 4210) a paragraph. In some embodiments, the
paragraphs in the document portion are selected in order, starting at the top-left, whereas in other
embodiments the paragraphs are selected in a random order.

Next, the process calculates (at 4215) the in-order, out-order, left-order, and right-
order, as well as sets of paragraphs that accompany each of these values. The out-order of a
paragraph \( p \) is calculated in some embodiments by using a set \( B(\cdot) \). The set \( B(\cdot) \) is initially all
paragraphs below paragraph \( p \) in the document portion that overlap \( p \) horizontally (i.e., that
overlap x-coordinates). For instance, Figure 43 illustrates a page 4300 with eleven paragraphs
including paragraph P 4305. The set B(P) is initially {Q, R, S, T, U}. Next, the paragraph closest to \( p \) is identified as \( q \), and all paragraphs that overlap paragraph \( q \) horizontally are removed from the set B(P). In the case of paragraph P 4305, paragraph Q 4310 is the closest to paragraph P, and paragraphs R 4315, S 4320, T 4325, and U 4330 are removed from the set B(P). At this point, the set B(P) is \( \{Q\} \).

[00313] Some embodiments then continue onto the next closest paragraph to \( p \) that was initially in the set \( B(p) \), and remove any paragraphs from \( B(p) \) that are below and horizontally overlap this next closest paragraph. Other embodiments continue to the next closest paragraph to \( p \) that remains in the set \( B(\ell) \), and remove any paragraphs from \( B(\ell) \) that horizontally overlap this paragraph. Either way, in the example of Figure 43, the set B(P) for paragraph P 4305 is \( \{Q\} \). The out-order of \( p \) is then the cardinality (i.e., number of elements) of the set B(\( \ell \)). This is repeated for each paragraph in B(\( \ell \)). Thus, in this case the out-order of paragraph P 4305 is 1. As an example of a paragraph with an out-order greater than 1, for paragraph R 4315, the set B(R) is \( \{S, X\} \), so that the out-order of paragraph R 4315 is 2.

[00314] The in-order of a paragraph \( p \) is calculated similarly to the out-order in some embodiments by using a set A(\( \ell \)). The set A(P) is initially all of the paragraphs in the document portion above \( p \) that overlap \( p \) horizontally. The closest paragraph to \( p \) is selected as paragraph \( q \), and the paragraphs that overlap paragraph \( q \) horizontally are removed from A(P). This is then repeated for each of the paragraphs in A(P). In the example page 4300, the set A(P) for paragraph P 4305 is the empty set, while the set A(R) for paragraph R 4315 is \( \{Q, W\} \). The in-order of a paragraph \( p \) is the cardinality (i.e., number of elements) of the set A(\( \ell \)).

[00315] The left-order and right-order of a paragraph \( p \) are also calculated similarly in some embodiments, using a set L(\( \ell \)) (paragraphs left of \( p \) and vertically overlapping \( p \), using the same removal rules) and a set R(\( \ell \)) (paragraphs right of \( p \) and vertically overlapping \( p \), using the same removal rules). Some embodiments use L(\( \ell \)) and R(\( \ell \)) for flow graphs (see below) when it has been determined (e.g., by an external means) that the language direction is top-down. For page 4300, the set R(P) for paragraph P 4305 is \( \{V\} \), while the set L(V) for paragraph V 4335 is \( \{P\} \). The sets L(R) and R(R) for paragraph R 4315 are both empty.

[00316] Once the in-order, out-order, left-order, and right-order are calculated for the selected paragraph, the process 4200 determines (at 4220) whether more paragraphs remain for
which the various values must be calculated. If more paragraphs remain, the process proceeds to
4210 to select another paragraph.

[00317] Otherwise, once the values are calculated for all paragraphs, the process generates
(at 4225) a flow graph for the paragraphs. The flow graph of some embodiments is generated
such that each paragraph in the document portion being evaluated is a node. A directed edge is
drawn from the node for a paragraph \( p \) to each node for the paragraphs in the set \( A(p) \). This is the
same, in some embodiments, as drawing a directed edge from each node for the paragraphs in the
set \( B(p) \) to the node for the paragraph \( p \). Figure 45 illustrates an initial flow graph 4501 for the
page 4300.

[00318] Next, process 4200 identifies (at 4230) call-outs. In some embodiments, identified
call-outs are removed from the flow graph. A call-out, in some embodiments, is a text element
on a page that is meant to be read in an order independent from the rest of the text on the page.
Some examples of call-outs include headers and footers, footnotes, margin notes, side-bars, and
other blocks of text placed amongst other elements such as large-font quotes in a magazine
article.

[00319] Some embodiments identify call-outs based on a combination of the geometry of
the text element, its position on the page, its flow properties (in-order, out-order, left-order, and
right-order), and the style properties of its elements. For instance, when a vertex \( v \) includes a
one-line paragraph that is close to the top of a page, the distance from the one-line paragraph to
any element in \( A(v) \) is more than one line height. \( L(v) \leq 1, R(v) \leq 1 \), and any vertices in \( L(v) \) and
\( R(v) \) share these conditions, then some embodiments classify the paragraph as a header call-out.
Requirements for a footer call-out arc similar in some embodiments, except looking for the
distance to the bottom of the page and to elements in \( B(v) \).

[00320] Some embodiments also identify sidebars that jut into columns (and are not in
their own zone), randomly located text boxes, small bits of text with no obvious relationship to
other text (e.g., figure captions), etc. as call-outs. Some embodiments make these determinations
(as well as other determinations of flow properties) based on a purely textual analysis, whereas
other embodiments incorporate images into the analysis (e.g., as further evidence for a figure
caption). For example, in some embodiments, some embodiments identify single-line paragraphs
distant from all elements in \( A(p) \) and \( B(p) \) as isolated small caption
Canrims are identified
in some embodiments when a paragraph with a single text line is enclosed by the bounds of an image and is aligned in particular ways with the image bounds (e.g., centered near the bottom, centered near the top, etc.).

[00321] When the rectangular bounding boxes of two or more paragraphs intersect, some embodiments identify all but one of the paragraphs as intersection call-outs. For instance, suppose that two paragraphs \( p \) and \( q \) overlap and \( B(/>) = \{ q, r \} \). When \( /r \) has an in-order of 1 or when \( q \) is in \( A(r) \), then \( q \) is an intersection call-out in some embodiments. Some embodiments classify as an intersection call-out any paragraph \( p \) whose style and/or alignment properties are not consistent with the paragraphs in \( A(p) \) or \( B(/r) \). When two paragraphs intersect, and none of the above rules applies, some embodiments classify the paragraph with smaller area as a call-out.

[00322] After generating the flow graph for the paragraphs in the document portion, the process 4200 merges (at 4235) nodes of the flow graph into columns. Some embodiments merge nodes for paragraphs \( /l \); and \( q \) if \( A(j/) = \{ q \} \) and \( B(q) = \{ l \} \). This indicates that paragraphs \( /l \); and \( q \) are in the same column in some embodiments. In some embodiments, the new node \( pq \) will have \( A(pq) = A(\langle y \rangle) \), \( B(pq) = B(/l) \), \( L(pq) = L(p) + L(q) \), and \( R(/xl) = R(/l) + R(q) \). For example, in Figure 45, the flow graph 4501 is modified such that nodes S 4520, T 4525, and U 4530 are merged into node STU 4575 in modified flow graph 4502. The other nodes are modified similarly.

[00323] Figure 46 illustrates a flow graph 4601 for the page 4400 of Figure 44 after the nodes have been merged initially into columns. Some embodiments identify paragraph R 4420 as a call-out because it straddles two columns and has paragraphs to both the left and right. Accordingly, some embodiments remove the node R 4620 from the flow graph 4601. This enables further merger of the nodes into columns.

[00324] Once call-outs have been identified (and, in some embodiments, removed from the flow graph), process 4200 partitions (at 4240) the flow graph into layouts. Some embodiments define labels for expansion and reduction edges as part of the partitioning process. In some embodiments, if the out-order of a paragraph \( /l \); is greater than 1, and the in-order of each paragraph \( q \) in the set \( B(/>) \) is 1, then the edge from \( p \) to each \( q \) in \( B(/>) \) is an expansion edge. Similarly, in some embodiments, if the in-order of a paragraph \( /l \); is greater than 1, and the out-order of each paragraph \( q \) in the set \( A(/;) \) is 1, then the edge from \( /l \); to each \( n \) is a...
reduction edge. Figure 45 illustrates that the edges leading into node R 4515 are both reduction edges, and the edges leading out of node R 4515 are both expansion edges.

[00325] The partitioning of some embodiments examines each vertex v the edges of which are all labeled. When the in-order of v is greater than 1, some embodiments define a partition the elements of which are BC(v) so long as A(p) = {v} for each p in B(v). Similarly, when the out-order of v is greater than 1, some embodiments define a partition the elements of which are A(v) so long as B(I) = {v} for each I in A(v). When both of these partitions are possible, the vertex v is defined as a partition by itself. Based on these rules, the flow graph 4502 is partitioned into three partitions 45 11, 45 12, and 45 13.

[00326] Some embodiments place any remaining nodes into one or more partitions such that the smallest number of partitions is defined without any geometric overlap between the partitions. Due to complex page structure, some embodiments use more relaxed partitioning rules than those described above. For instance, when a partition could be created from a node v, except that the out-order of v is greater than 1, then elements of A(v) that are far from v and narrow relative to v are eliminated in some embodiments. When only one element remains in A(v), the edges from v to the removed vertices are removed, and partitioning is continued. Once partitioning is complete, the process 4200 ends.

[00327] In some embodiments, each partition corresponds to a linear layout, and each of the final (merged) nodes corresponds to a column. Once partitions are defined, some embodiments calculate properties of the document portion such as gutter width, margins, in-line or floating images, etc.

[00328] Furthermore, layout and flow information (including word, line, paragraph, and column data) is used prominently in the display of the document and enabling more robust user interaction with the document, as described below in Sections VUI and IX. For instance, in some embodiments, a user might wish to view a complex document that includes several columns of text, images, call-outs, captions, etc., and be able to copy and paste the entire text of the document into a text editor. In order for this to be accomplished, a reading order is assigned to each of the elements in the document that attempts to identify the order in which a human would read through the elements of the document.
For instance, some embodiments assign reading orders to columns, such that the reading order follows the expected order in which a human would read the columns from the start to end of the document or page. Other embodiments assign reading orders to other structural elements (e.g., paragraphs, words, etc.). In some embodiments, when the user copies and pastes the entire text of such a document into another application, the text appears in the application in the order that a human would read it. This is in contrast to copying and pasting from a standard PDF file that orders all text in a strict top-down configuration.

Some embodiments also insert images and shapes into the reading order. For instance, some embodiments will identify a particular image as associated with a particular column of text and insert the image either before or after (depending on the evidence in the document) the column of text. As an example, some embodiments identify that an image is associated with the caption for the image and insert the image into the reading order immediately prior to its caption.

Some embodiments also define links between structural elements. For instance, some embodiments use the reading order to define links between a paragraph at the end of a column and a paragraph at the beginning of the next column that are actually one paragraph. In some embodiments, to maintain the hierarchy that has each paragraph assigned to one particular column, a separate paragraph bridging the columns is not defined. Instead, a link between the two paragraphs is defined indicating that they are, in fact, one paragraph. Some embodiments use tests similar to those for adding lines to a paragraph in order to determine whether the top paragraph from a second column is actually a continuation of the paragraph at the end of a first column (i.e., examining spacing, alignment, font stylings, etc.). The link can then be used, e.g., if a user performs a selection operation (e.g., a triple-click) intended to select a paragraph within either of the defined paragraphs, the entire actual paragraph will be selected based on the link.

Some embodiments also define links between layouts (e.g., linking across pages) or zones. For instance, some embodiments can recognize continuation text (e.g., text in a newspaper indicating that a story continues on a different page) and can link the text in the layout with the continuation text to the layout where the text continues. Some embodiments only attempt such linking when a profile has been matched indicating that linking should be performed. For instance, if a document has been identified as a newspaper, then some
embodiments will search for continuation text.

E. Software Architecture

In some embodiments, the layout and flow analysis processes described above are implemented as software running on a particular machine, such as a computer, a media player, a cell phone (e.g., an iPhone®), or other handheld or resource-limited devices (or stored in a computer readable medium). Figure 47 conceptually illustrates the software architecture of a layout and flow analysis application 4700 of some embodiments for identifying layout and flow characteristics of a document. In some embodiments, the application is a stand-alone application or is integrated into another application (e.g., a document reconstruction application), while in other embodiments the application might be implemented within an operating system.

Layout and flow analysis application 4700 includes a line identification module 4705, a line-merging module 4710, a word identification module 4715, a difference clustering module 4720, a line splitting module 4725, a paragraph identification module 4730, a column and layout identification module 4735, and an order calculator 4740.

Figure 47 also illustrates document content 4745. Line identification module 4705 receives information from the document content 4730. In some embodiments, this information is information about the position of characters in the document. Line identification module 4705 identifies characters with a common baseline on a page and assigns them to a line. The line identification module passes information to, and receives information from, line merging module 4710. The line merging module identifies groups of lines that overlap vertically and determines whether the lines should be merged. In some embodiments, line merging module 4710 performs some or all of process 3300 described above. The line merging module 4710 passes this information back to line identification module 4705, which identifies the final text lines. Line identification module 4705 passes the line information back to the document content 4745, as well as to line splitting module 4725.

Word identification module 4715 also receives information from the document content 4745. In some embodiments, this information is information about the position of characters in the document. The word identification module 4715 identifies characters that should be grouped together as words. Word identification module 4715 passes information to,
and receives information from, the difference clustering module 4720. Difference clustering module 4720 performs difference clustering on the document characters to return different levels of gaps between characters (e.g., word gaps, segment gaps, etc.). The word identification module 4715 uses the difference clustering results to identify the words. Word identification module 4715 passes its results (as well as other difference clustering results such as segment gaps) to the document content 4745, as well as to line splitting module 4725.

[00337] Line splitting module 4725 receives line information from the line identification module and gap information from the word identification module, as well as other information (e.g., gutter information) from the document content 4745. Line splitting module 4725 identifies where lines should be split and outputs new line information based on the splits. The new line information is passed to document content 4745 as well as paragraph identification module 4745. In some embodiments, line splitting module 4725 performs some or all of process 3800.

[00338] Paragraph identification module 4730 receives line information from line splitting module 4725 as well as other information (e.g., alignment information) from document content 4745. Paragraph identification module 4730 identifies which lines should be grouped into paragraphs and outputs the result information. The paragraph information is passed to document content 4745 as well as to the column and layout identification module 4735. In some embodiments, paragraph identification module 4730 performs some or all of process 4000.

[00339] Column and layout identification module 4735 receives paragraph information from paragraph identification module 4730, as well as other information (e.g., zone information) from document content 4745. Column and layout identification module 4735 groups paragraphs into columns and groups columns into layouts. Column and layout information module 4735 passes information to, and receives information from, order calculator 4740. The order calculator 4740 receives paragraph information from the module 4735, and calculates the in-order, out-order, left-order, and right-order (as well as the corresponding sets A, B, L, and R) for the paragraphs. This information is then returned to the module 4735 for use in generating a flow graph. The results from column and layout identification module 4735 are passed to the document content 4745. In some embodiments, column and layout identification module 4745 performs some or all of process 4200 described above.

[00340] In some embodiments, the results of the paragraph, column, and layout identification are passed to the document content 4745.
described modules or other modules are stored in an electronic storage (e.g., as part of a
document object model). The document object model can then be used for displaying the
document on an electronic display device (e.g., a handheld device, computer screen, etc.) such
that a user can review and/or interact with the document (e.g., via touchscreen, cursor control
device, etc.).

V. TABLE IDENTIFICATION

[00341] Some embodiments of the invention identify tables in a document. Some
embodiments identify cells, rows, and columns for the tables and enable the tables for editing,
export to a spreadsheet, etc. Zone information from processes described above in Section II, as
well as layout and flow information from processes described above in Section IV, is used in the
table identification process of some embodiments. Some embodiments can identify and
reconstruct tables even when the tables are drawn as an unrelated group of graphics included
overlapping shapes and images, such as is common when the table includes effects such as
extrusions and shadows.

[00342] Figure 48 conceptually illustrates a process 4800 of some embodiments for
identifying tables in a portion of a document. As shown, the process receives (at 4805) zone and
layout information for a document portion. In some embodiments, the zone information is the
output of processes described above in Section II, and the layout information is the output of
processes described above in Section IV. In some embodiments, the document portion is a page
of a document, or a zone of a page, etc.

[00343] The process then identifies (at 4810) tables with complete borders. Figure 49
illustrates an example of a table 4900 with complete borders. Some embodiments classify any
zone that is an island (as defined above in Section U) partitioned entirely by two or more zones
whose bounds are all upright rectangles. These partitioning rectangles are thus the cells of the
table. The identification of tables with complete borders is described in further detail in
subsection A below.

[00344] Process 4800 next identifies (at 4815) tables with connected borders. Figure 50
illustrates an example of a table 5000 with connected, but incomplete borders. A table has
connected borders in some embodiments when all of the borders of the table that are actually
drawn intersect each other such that any border is connected to any other border through a set of connections. Table 5000 has incomplete borders because the outer borders of the table arc not drawn. The identification of tables with connected but incomplete borders is described in further detail in subsection B below.

[00345] Process 4800 next identifies (at 4820) tables with no borders or disconnected borders. Figure 51 illustrates an example of a table 5100 with no borders. Figure 52 illustrates a table 5200 with disconnected borders. Table 5200, unlike table 5100, has some borders drawn, but the borders do not form a connected set. For tables whose borders do not form a connected set, some embodiments require the use of layout information in order to identify the tables. The identification of tables with no borders or disconnected borders is described in further detail in subsection C below.

[00346] Once all of the tables in the portion of the document arc identified, the process defines (at 4825) the tables to be accessible by editing and other software. The process then ends. In some embodiments, this enables a user to edit individual cells of the table, select rows and columns of the table, copy the table information intelligently into a spreadsheet, etc.

A. Tables with Complete Borders

[00347] Some embodiments of the invention identify tables in which all of the borders are drawn with some combination of border graphics (e.g., lines, narrow rectangles, narrow images, borders of filled rectangles, etc.). Processes of some embodiments for identifying these border graphics and using the border graphics to identify zones arc described above in Section II.

[00348] Figure 53 conceptually illustrates a process 5300 of some embodiments for identifying tables that have a complete set of borders and defining the cells, rows, and columns of the identified tables. In some embodiments, process 5300 is also used to identify and define tables with incomplete or no borders once implied borders have been identified for those tables. Such processes are described in further detail in subsections B and C. Process 5300 will be described in conjunction with Figure 54. Figure 54 illustrates a page 5400 that includes a table with complete borders.

[00349] As shown in Figure 53, the process receives (at 5305) zone information for a portion of a document. In some embodiments, the portion of the document is the entire
document, a section of the document, or a page of the document. The zone information in some embodiments is information output from processes described above in Section II and includes zone border and intersection information as well as the identified zones and zone graph information.

[00350] The process 5300 then identifies (at 5310) islands partitioned entirely by two or more zones, all of which are rectangular. As described above, in some embodiments, an island is a zone that does not share a border interval with its parent zone. Figure 54 illustrates a page 5400 that includes one island 5401 partitioned by rectangles. As shown, the thirteen rectangles completely partition the island 5401 in that the region of the page covered by the thirteen rectangular zones is exactly that of the island 5401. Accordingly, island 5401 is identified as a table.

[00351] Next, the process determines (at 5311) whether there are any unprocessed islands of those identified at 5310. When none were identified or all that were identified have been processed as tables, the process ends. Otherwise, the process selects (at 5314) the next identified island. In some embodiments, there is no particular ordering to the identified islands, so long as all of them are processed.

[00352] Process 5300 then sorts (at 5315) the horizontal borders in the selected island from highest to lowest. Some embodiments sort by the top boundary of the border such that the border with the highest top bound is first. The horizontal borders in some embodiments are the horizontal border intervals within the island identified as a table and including the borders of the island. Figure 54 illustrates sixteen horizontal border intervals 5405-5420. In some embodiments, the horizontal border intervals are identified by a process such as process 900 described above in Section II.

[00353] With the horizontal border intervals sorted in order, the process selects (at 5320) the first unevaluated horizontal border and defines (at 5325) the next horizontal gridline of the table. In the example of Figure 54, either border interval 5405 or 5406 is the first selected border, at which point the horizontal gridline 1 is defined.

[00354] The process then assigns (at 5330) the selected border to the current gridline and sets the vertical extent of the gridline to the intersection of y-intervals of all borders assigned to
the gridline. Thus, when the currently selected border is the first to be assigned to the current gridline, the \( y \)-interval of the gridline is simply the \( y \)-interval of the border. When multiple borders are assigned to the gridline, however, the \( y \)-interval of the gridline includes the \( y \)-intervals of all of the assigned borders.

[00355] The process then determines (at 5335) whether there are more horizontal borders that have not been assigned to a gridline. When there are no more borders horizontal borders remain, the process proceeds to 5345 which is described below. Otherwise, when there are more borders remaining, the process determines (at 5340) whether the \( y \)-interval of the next border overlaps the \( y \)-interval of the current gridline. When the \( y \)-intervals do not overlap, all of the borders in the current gridline have been defined, and the process proceeds to 5320 to select the next border and begin a new gridline. However, when the \( y \)-intervals overlap, the process proceeds to 5330 to assign the border to the current gridline, update the \( y \)-interval of the gridline, and continue to the next border.

[00356] As shown in Figure 54, after either selecting border 5405 or 5406 as the first border for island 5401, the other border of those two would be selected next and added to horizontal gridline 1. Horizontal gridline 2 would next be defined, including borders 5407 and 5408, and so on down to horizontal gridline 6 (borders 5418-5420).

[00357] Once all of the horizontal borders are assigned to gridlines, process 5300 considers the vertical borders. Process 5300 sorts (at 5345) the vertical borders in the selected island from left to right. Some embodiments sort by the left boundary of the border such that the border with the leftmost left bound is first. The vertical borders in some embodiments are the vertical border intervals within the island identified as a table and including the borders of the island. Figure 54 illustrates eighteen vertical border intervals 5425-5442. In some embodiments, the vertical border intervals are identified by a process such as process 900 described above in Section II.

[00358] With the vertical border intervals sorted in order, the process selects (at 5350) the first unevaluated vertical border and defines (at 5355) the next vertical gridline of the table. In the example of Figure 54, one of border intervals 5426-5428 is the first selected border, at which point the vertical gridline 1 is defined.
The process then assigns (at 5360) the selected border to the current gridline and sets the horizontal extent of the gridline to the intersection of x-intervals of all borders assigned to the gridline. Thus, if the currently selected border is the first to be assigned to the current gridline, the x-interval of the gridline is simply the x-interval of the border. When multiple borders are assigned to the gridline, however, the x-interval of the gridline includes the x-intervals of all of the assigned borders.

The process then determines (at 5365) whether there are more vertical borders that have not been assigned to a gridline. When there are no more borders remaining, the process proceeds to 5375 which is described below. Otherwise, when there are more borders remaining, the process determines (at 5370) whether the x-interval of the next border overlaps the x-interval of the current gridline. When the x-intervals do not overlap, all of the borders in the current gridline have been defined, and the process proceeds to 5350 to select the next border and begin a new gridline. Otherwise, when the x-intervals overlap, the process proceeds to 5360 to assign the border to the current gridline, update the x-interval of the gridline, and continue to the next border. After selecting one of the borders 5425-542K as the first border for island 5401, one of the other three are selected and added to vertical gridline 1. Once all four of these borders are added to vertical gridline 1, vertical gridline 2 is next defined, including borders 5429-5433, and so on to vertical gridline 4 (borders 5438-5442).

Once all of the vertical borders are evaluated, all gridlines have been defined for the table. One of ordinary skill in the art would recognize that while process 5300 defines horizontal gridlines prior to vertical gridlines, some embodiments define vertical gridlines first.

The process next assigns (at 5375) row and column numbers to the cells of the table. Each of the zones that partitions the island is a cell in some embodiments. In some embodiments, a cell spans the rows from its upper boundary's gridline to one less than its lower boundary's gridline. For example, cell 5450 in Figure 54 spans rows 1 and 2, because its upper border is part of horizontal gridline 1 and its lower border is part of horizontal gridline 3. Similarly, in some embodiments, a cell spans the rows from its left boundary's gridline to one less than its right boundary's gridline. For example, cell 5450 spans column 1, because its left border is part of vertical gridline 1 and its right border is part of vertical gridline 2.

Once the table Structure (i.e., the gridlines relk row and column) have been
reconstructed, the process determines (at 5380) styling information for the table. The process then proceeds to 5311, described above, to determine whether there are any more identified islands to process as tables. In some embodiments, the table styling information comes from the border graphics that make up the zone borders as well as background graphics. For example, when there are multiple background shapes covering or crossing a particular cell, some embodiments determine the resulting shape color or image rendering by compositing the various background shapes, and clipping it to the gridline boundary if necessary. Similarly, some embodiments determine cell border colors based on a weighted average of shape colors that show through. The weights in the weighted average arc be based on the amount of area showing for each color in some embodiments. Some embodiments also recognize border stylings such as shadows or extrusions.

[00364] In some embodiments, identifying a table and determining the table structure enables the table to be utilized as if generated by a typical word processor rather than a set of unrelated graphics. For example, in some embodiments, users can edit the table cells individually, export the information to a spreadsheet, sort the table data, etc.

[00365] Some embodiments only use process 5300 to identify tables. Doing so will only identify tables with a complete set of borders, however. Other embodiments use processes described in subsections B and C, below, to identify tables that do not have complete border graphics, but otherwise have the structure and layout of a table.

B. Tables with Connected Borders

[00366] In addition to tables that have a complete set of borders, some embodiments identify tables that have an incomplete, but connected, set of borders. Figure 50 illustrates such a table 5000. Some embodiments use processes to define implied borders that could be intended table borders, then apply a process such as process 5300 described above to identify and reconstruct the actual table.

[00367] Figure 55 conceptually illustrates a process 5500 of some embodiments for identifying and reconstructing tables that have a connected set of borders. Process 5500 will be described in conjunction with Figure 56. Figure 56 illustrates a sequence of identifying a table with connected but incomplete borders on a page 5600.
As shown in **Figure 55**, process 5500 receives (at 5505) a set of potential zone borders and border intersections for a portion of a document. In some embodiments, the potential zone borders and intersections are determined by process 900 as described above in Section II. Other embodiments use other processes to determine the potential zone borders for a document portion. The document portion in some embodiments is the entire document, a section of the document, or a page of the document.

Next, process 5500 defines (at 5510) a set $U$ that includes all of the potential zone borders received at 5505. The process then determines (at 5515) whether $U$ is empty. When the set is empty, the process ends because there are no zone borders that could be connected.

Otherwise, when the set $U$ includes at least one zone border, the process selects (at 5520) a border $b$ from $U$ and defines a new connected set $C$. Some embodiments start with the border that is topmost, leftmost, closest to the top left corner, or use some other heuristic to determine which border to select first. Other embodiments select the border $h$ randomly. The selected border $b$ is then removed (at 5525) from the set $U$ and added to the set $C$. The connected set $C$, in some embodiments, is a set of borders that are all connected through a set of intersections.

The process then selects (at 5530) a border $z$ from the set $U$ that has not yet been evaluated for the current border $b$, and determines (at 5535) whether the border $z$ intersects the border $h$. In some embodiments, determining whether border $z$ intersects border $b$ involves determining whether one of the received intersections connects border $z$ to border $b$. When the borders do not intersect, the process proceeds to 5545 which is described below. Otherwise, when the borders intersect, the process adds (at 5540) the border $z$ to the current connected set $C$ and places $z$ in a queue. Some embodiments use the queue to keep track of borders in a connected set that need to be evaluated for further connections before moving on to the next connected set.

Next, the process determines (at 5545) whether any borders in the set $U$ have not yet been evaluated for intersection with the current border $b$. When more borders remain, the process proceeds to 5530 to select another border $z$ and determine whether the new border $z$ intersects the current border $z$. 


Otherwise, when all borders in $U$ have been evaluated for intersection with the current border $h$, the process 5500 determines (at 5550) whether the queue is empty. When there is at least one border in the queue, the process sets (at 5555) the next border in the queue as the new selected border $h$. The process then proceeds to 5525 to evaluate whether the borders remaining in $U$ intersect the newly selected border $h$.

Otherwise, when the queue is empty, all elements of the current connected set $C$ have been assigned and the process determines (at 5560) whether the set $U$ is empty. When $U$ is not empty, then the process proceeds to 5520 to select a new border $h$ and define a new connected set $C$.

When $U$ is empty, then all of the borders have been processed and placed in their respective connected sets. Figure 56 illustrates a page 5600 with seven borders 5605-5635. Applying process 5500, these seven borders are assigned to three connected sets 5640 (including border 5605), 5645 (including borders 5610 and 5615) and 5650 (including borders 5620-5635). Each of these connected sets includes borders that intersect at least one other border in the connected set and do not intersect any borders outside of the connected set.

Once all of the connected sets are identified because the set $U$ is empty, the process selects (at 5565) a set $C$, and adds (at 5570) to the set $C$ the four borders that form the upright bounding box of $C$. The upright bounding box, in some embodiments, is the smallest upright rectangle that includes all of the borders in the connected set. For instance, Figure 56 illustrates the four borders 5655-5670 that form the upright bounding box of connected set 5650. Connected set 5640 is a single line, so its upright bounding box is just the outer edges of the line. Two of the four borders that form the upright bounding box of connected set 5645 are the borders 5610 and 5615, while the additional borders 5675 and 5680 are added by operation 5570 in some embodiments.

The process then determines (at 5575) whether any connected sets $C$ remain. When at least one set remains, the process proceeds to 5565 to select another set $C$. When all sets have been evaluated, the process identifies (at 5580) zones using all of the borders from all of the connected sets. Some embodiments apply process 1200 (described above in Section II) or a similar process to identify the zones.
Once the zones are identified, process 5500 identifies (at 5585) tables from the zones and reconstructs the tables. The process then ends. Some embodiments apply process 5300, as described above in subsection A, to reconstruct the tables once the zones are identified using the implied borders from the upright bounding boxes of connected sets. **Figure 56** illustrates that one table 5685 is identified on page 5600. The table has three columns and three rows, with nine total cells.

Some embodiments then remove the table and cell zones (and the border graphics that make up these zones) and re-identify zones without any implied borders so as to not define zones where there should not be any. Some embodiments then re-insert the table and cell zones to determine the zone graph as described above in Section II. Such embodiments generate a full zone graph that includes identification of table and cell zones and their structure.

### C. Tables with No Borders or Disconnected Borders

In addition to tables that have a complete or at least a connected set of borders, some embodiments identify tables that have either no borders or disconnected borders. **Figure 51** illustrates a table 5100 with no borders, while **Figure 52** illustrates a table 5200 with disconnected borders. Some embodiments use processes to define implied borders that could be intended table borders, then apply a process such as process 5300 to identify and reconstruct the actual table. For tables whose borders do not form a connected set, some embodiments require the use of layout information in order to identify the tables in addition to zone information.

**Figure 57** conceptually illustrates a process 5700 for identifying and reconstructing tables with no borders or disconnected borders. Process 5700 will be described in conjunction with **Figures 58-60. Figures 58-60** illustrate the application of process 5700 to a page 5800 to identify a table that has disconnected borders.

As shown in **Figure 57**, the process receives (at 5705) layout information for a primary zone. In some embodiments, the primary zone is a page, though the primary zone can be any other zone as well. The layout information of some embodiments is information output from processes such as 4000 and 4200, described above in Section IV. This information includes identification of paragraphs, columns, and layouts in the primary zone.

Process 5700 then determines (at 5707) whether there are any unprocessed
layouts in the primary zone. When there are no layouts in the primary zone, or all layouts have been processed, the process proceeds to 5740, described below. Otherwise, the process selects (at 5710) a layout within the primary zone. The process then defines (at 5715) implied zone borders at the upright rectilinear bounds of the layout. In some embodiments, the upright rectilinear bounds is the smallest rectangle that completely encloses all paragraphs in the layout. The process also defines (at 5720) implied zone borders at the vertical white space separating columns of the layout and horizontal white space separating paragraphs of the layout within a column.

[00384] Figure 58 illustrates a page 5800. The page includes three smaller zones: zone 5805 in the upper left corner, zone 5810 shaded gray in the center, and zone 5815 towards the bottom. Zones 5810 and 5815 are islands in some embodiments. The primary zone (page 5800) includes two layouts 5820 and 5825. As illustrated in Figure 59, the process, having selected layout 5820, defines implied borders 5905-5925 around the bounding box of layout 5820 and between the three columns of layout 5820. The bottom bound of the layout does not need an implied border in some embodiments because the layout is bounded by the top of zone 5810. Similar implied borders 5985-5997 are defined for layout 5825.

[00385] Next, process 5700 determines (at 5725) whether there is any horizontal border (e.g., implied border) in the layout which, extended outward, intersects the bounding box of a paragraph in another column. This horizontal consistency test ensures that the layout actually has a tabular structure, as opposed to just being a typical set of paragraphs that are broken into columns. When the layout does not pass the horizontal consistency test, the process proceeds to 5707 which is described above. Otherwise, when the layout passes the horizontal consistency test, then the process adds (at 5730) the layout to a set 5 that keeps track of potential layouts that could be a table or part of a table. The process then proceeds to 5707.

[00386] After the process determines (at 5707) that the primary zone includes no more layouts, the process determines (at 5740) whether there are any unprocessed zones within the primary zone. When the primary zone includes no other zone, or all zones have been processed, the process is finished defining implied borders and proceeds to 5775 which is described below.

[00387] Otherwise, when there is at least one unprocessed zone in the primary zone, the process selects (at 5745) one of the smaller zones. The process then determines (at 5750)
whether the smaller zone is an island with only one layout. On page 5800, zones 5810 and 5815 are islands with only one layout, whereas zone 5805 includes only one layout but is not an island because its borders include page borders.

[00388] When the selected zone is not an island with only one layout, the process 5700 proceeds to 5740 which is described above. Otherwise, when the selected zone is an island with only one layout, the process defines (at 5755) implied zone borders at the vertical white space separating columns of the layout and horizontal white space separating paragraphs of the layout within a column. Figure 59 illustrates implied zone borders 5971 and 5972 that are defined between the columns of zone 5810.

[00389] The process then determines (at 5760) whether there is any horizontal border (e.g., implied border) in the layout which, extended outward, intersects the bounding box of a paragraph in another column. This horizontal consistency test ensures that the layout actually has a tabular structure, as opposed to just being a typical set of paragraphs that are broken into columns. When the layout does not pass the horizontal consistency test, the process proceeds to 5740 which is described above. Otherwise, when the layout passes the horizontal consistency test, the process adds (at 5765) the layout to a set S that keeps track of potential layouts that could be a table or part of a table, then proceeds to 5740.

[00390] Page 5800 includes zone 5815, which in some embodiments is an island with only one layout. Figure 59 illustrates the implied borders 5930-5980 defined for zone 5815. However, this layout does not pass the horizontal consistency test because the horizontal borders 5945-5980 would all intersect the bounding box of other paragraphs in the layout if extended outwards. Accordingly, the layout of zone 5815 is not added to the set S for page 5800, and the implied borders 5930-5980 are not kept.

[00391] Once all the zones have been evaluated, the process applies (at 5775) zone analysis using the implied zone borders from the layouts of the set S as well as any zone border graphics that intersect these implied zone borders. Some embodiments use the processes described above in Section II in order to identify zone border intervals, intersections, and zone borders.

[00392] Process 5700 then identifies (at 5780) potential tables among the layouts of the
set $S$. Some embodiments use process 5300, described above in subsection A, to identify potential tables (and reconstruct their structure). The process next disqualifies (at 5785) potential tables that fail to meet specific table conditions. The process then ends. Some embodiments impose additional conditions on potential tables to ensure that there exists strong evidence for identifying a table when the border graphics alone are insufficient. For example, some embodiments require that all cells span exactly one row and one column, or that the cell height and width be small compared to the page dimensions, or that the thickness of each border is small compared to the dimensions of the table. Some embodiments require one, some, or all of these conditions, as well as other requirements. Figure 60 illustrates that zone 5810 and layouts 5820 and 5825 are combined into table 6000, while the content of zones 5805 and 5815 remains as is.

Once all tables in a document portion are identified, in some embodiments users are able to edit individual cells of the table, select rows and columns of the table, copy the table information intelligently into a spreadsheet, etc. Furthermore, the text flow, reading order, and display can be improved in some embodiments by identifying tables and reconstructing them as such.

D. Software Architecture

In some embodiments, the table identification described above are implemented as software running on a particular machine, such as a computer, a media player, a cell phone (e.g., an iPhone®), or other handheld or resource-limited devices (or stored in a computer readable medium). Figure 61 conceptually illustrates the software architecture of a table identification application 6100 of some embodiments for identifying tables in a document. In some embodiments, the application is a stand-alone application or is integrated into another application (e.g., a document reconstruction application), while in other embodiments the application might be implemented within an operating system.

Table identification application 6100 includes a zone analysis module 6105, a table identification module 6110, a zone border connection module 6115, an implied border definition module 6120, and a layout analysis module 6125.

Figure 61 also illustrates document content 6130. Zone analysis module 6105
receives information (e.g., information about graphics) from document content 6130. In some
eembodiments, zone analysis module 6105 is the same as zone analysis module 1700 from Figure
17. The zone analysis module outputs zone information to the zone border connection module
6115 and the table identification module 6110.

[00397] Table identification module 6110 receives zone information from the zone
analysis module 6105. In some cases, these are the final zones of a document, while in other
cases, the zone information is zones that include implied borders for the specific purpose of table
identification. Table identification module 6110 identifies tables based on the zone information
received, and passes this information to the document content 6145. In some embodiments, table
identification module 6110 performs some or all of process 5300.

[00398] Zone border connection module 6115 receives zone border information from the
zone analysis module 6105, and defines connected sets of zone borders based on this
information. The connected sets are passed to implied border definition module 6120. In some
embodiments, zone border connection module 6115 performs some or all of process 5500.

[00399] The layout analysis module 6125 receives information (e.g., layout information)
from the document content 6130. The layout analysis module 6125 determines whether layouts
could potentially be part of a table, and passes qualifying layouts to the implied border definition
module 6120. In some embodiments, layout analysis module 6125 performs some or all of
process 5700.

[00400] The implied border definition module 6120 receives layout information from the
layout analysis module 6125 and connected sets of zone borders from the zone border
connection module 6115. The implied border definition module 6120 defines implied borders for
a page of a document based on the information it receives, and passes this information to zone
analysis module 6105.

[00401] In some embodiments, the results of the processes performed by the above-
described modules or other modules are stored in an electronic storage (e.g., as part of a
document object model). The document object model can then be used for displaying the
document on an electronic display device (e.g., a handheld device, computer screen, etc.) such
that a user can review and/or interact with the document (e.g., via touchscreen, cursor control
VI. JOINING GRAPHS

[00402] In some embodiments, unstructured document will include primitive elements (e.g., shapes and images) that are intended to be treated as a single element but are not defined as such in the document. When such primitive elements occupy a compact and isolated area of a document, they can be associated using a novel cluster analysis technique referred to as bounds clustering. The objective of bounds clustering, in some embodiments, is to minimize the spread of a cluster, where the spread is calculated from the bounds of the collection of primitive elements (e.g., shapes) in the cluster, while simultaneously maximizing the number of primitive elements in the cluster. The bounds, in some embodiments, are based on the bounding boxes for a shape or collection of shapes.

[00403] Some embodiments of the invention provide methods for identifying graphs (i.e., graphic objects) of a region that should be joined. These joined graphs can then be treated as one object for the purposes of further reconstruction. Furthermore, they can be treated as one object when viewed, selected, zoomed, copied, moved, edited, etc. Some embodiments treat joined graphs as one object for use in selection, display, and navigation processes described below in Sections VIII and IX.

[00404] Figure 62 conceptually illustrates a process 6200 of some embodiments for joining individual graphs into joined graphs. Process 6200 will be described in conjunction with Figure 63. Figure 63 illustrates the joining of some, though not all, of several graphs on a page 6300. As shown in Figure 62, process 6200 receives (at 6205) a portion of a document. The document portion is an entire document, a section of a document, a page, or a zone in some embodiments. Some embodiments perform the graph joining process for the entire document at once, while some embodiments perform the process on a zone-by-zone or page-by-page basis.

[00405] The process identifies (at 6210) graphs in the document portion. Figure 63 illustrates a page 6300 that includes six graphs: a seven-pointed star 6305, a pentagon 6310, an octagon 6315, a cross 6320, a triangle 6325, and a five-pointed star 6330.

[00406] The process then uses cluster analysis to join (at 6215) some of the identified graphs. The process then ends. Some embodiments use a form of cluster analysis called bounds
clustering that is described in detail below by reference to process 6400. Some embodiments apply efficiency techniques described below in Section X to perform the cluster analysis. Some embodiments only join graphs when they are close together and do not take up too large a portion of a page or zone. Figure 63 illustrates that seven-pointed star 6305 and pentagon 6310 are joined into a single graph 6335, and triangle 6325 and five-pointed star 6330 are joined into a single graph 6340. Because they are isolated on page 6300, octagon 6315 and cross 6320 are not joined either to each other or to any other graphs.

A. Bounds Clustering

Figure 64 conceptually illustrates a process 6400 of some embodiments for performing bounds clustering to identify graphs that should be joined and joining those graphs. In some embodiments, process 6400 takes advantage of memory and processing efficiencies described below in Section X (e.g., indirectly sorted arrays, quick partitioning, etc.). As shown, the process receives (at 6405) graphs for a document portion. The document portion is an entire document, a section of a document, a page, or a zone in some embodiments.

The process then determines (at 6407) whether there are at least two graphs in the document portion. When there are one or zero graphs, there is no reason to perform clustering to attempt to join graphs, therefore the process ends. Otherwise, the process sorts (at 6410) the graphs by drawing order. The drawing order, in some embodiments, is the sequence in which objects are drawn on a page. Often, when multiple objects are intended to be treated as a single object, they will be drawn in sequence. Some embodiments, however, sort based on other heuristics, such as the location of the object on the page.

Next the process sets (at 6415) the first graph in the drawing order as the current graph g. The process then determines (at 6420) whether g is the last graph in the document portion. When g is the last graph, then no spread between graph g and a next graph can be calculated, so the process proceeds to 6440 which is described below.

Otherwise, when the graph g is not the last graph, the process calculates (at 6425) a spread between the graph g and the next graph in the drawing order, and stores (at 6430) the calculated spread in an array. A spread, in some embodiments, is a measure of how close together two objects are to each other. Some embodiments use the bounding boxes of the two objects.
objects to calculate the spread. For example, some embodiments calculate the spread of a set of
graphic objects as the sum of the width and the height of the smallest upright bounding box
into which the set of objects fits, divided by the sum of the width and height of the page.

Figure 65 illustrates two pages 6501 and 6502, each having two graphic objects
for which the spread is calculated. Page 6501 includes two graphic objects 6505 and 6510, while
page 6502 also includes two graphic objects 6515 and 6520 having the same shapes and sizes as
objects 6505 and 6510, but located at different places on the page. Figure 65 also illustrates the
smallest bounding box 6525 for objects 6505 and 6510 and the smallest bounding box 6530 for
objects 6515 and 6520. Using the metric to calculate spread mentioned above, the spread for
objects 6505 and 6510 is \((X_{s1} + Y_{s1}V(Y_{p} + X_{p}))\), while the spread for objects 6515 and 6520 is
\((X_{s2} + Y_{s2}V(Y_{p} + X_{p}))\). Some embodiments instead calculate the spread as the area of the bounding
box for the collection of objects divided by the area of the page. Some embodiments use metrics
that do not relate to the page size such as the size of the bounding box for the collection of
objects compared to the individual bounding boxes of the objects themselves.

Next, the process sets (at 6435) the next graph as the current graph \(g\). The process
then proceeds to 6420 which was described above. Once all the spreads have been calculated, the
process uses (at 6440) the spreads as first-order differences for difference clustering in order to
define clusters of graphs. Some embodiments perform difference clustering as described in the
above mentioned United States Publication No. 2007/0250497. As difference clustering of some
embodiments only requires the differences between the input values, and does not require the
actual values of the inputs, the spreads can be used as the first-order differences despite not
arising as actual differences. Clusters that result from difference clustering will, in some
embodiments, have relatively small spreads between consecutive graphs in the same cluster as
compared to the spreads between graphs in different clusters.

One of ordinary skill in the art would recognize that the spread, and thus the
concept of bounds clustering, is not limited to graphic objects on a page. For example, spreads
can be calculated among three-dimensional objects (by using volumes rather than areas or by
summing over the bounding boxes in three dimensions rather than two), and thus be used to
cluster three-dimensional objects (e.g., in a three-dimensional media-editing application such as
a video compositing application).
After difference clustering is used, with the spreads as first-order differences, clusters of graphics are defined. Process 6400 selects (at 6445) a cluster C from the unevaluated clusters. The process then processes (at 6450) C into a set of subsequences of graphs that meet certain constraints. Different embodiments use different constraints to define the joined graphs.

Some embodiments impose the requirement that the objects in a subsequence must be consecutive in drawing order. Some embodiments require that the objects in a sequence be mutually overlapping in that there is no way to partition the cluster into two nonempty subsequences, each of which is consecutive in drawing order, such that the upright bounds of the group of objects in the first partition is disjoint from the upright bounds of the group of objects in the second partition. A third requirement imposed by some embodiments is that each subsequence meets density constraints, which ensure that each subsequence includes a sufficient number of graphs (e.g., two) with a sufficiently small total spread.

Some embodiments use modified versions of the above conditions. For example, instead of the upright rectangular bounds, some embodiments use tighter bounds such as a path around the non-transparent pixels of an image. In some embodiments, the collection of objects in each of these subsequences is joined as a single graph.

Process 6400 next determines (at 6455) whether there are more clusters to evaluate. When more clusters remain, the process proceeds to 6445 to select another cluster and process that cluster into subsequences. Otherwise, when all clusters have been processed, the process ends. Now that the graphs are joined, they can be treated as one object when viewed, selected, zoomed, copied, moved, edited, etc. Some embodiments treat joined graphs as one object for use in selection, display, and navigation processes described below in Section VIII.

B. Processing Clusters into Subsequences

As noted above, after clusters of graphs have been identified, some embodiments process each cluster into subsequences to identify the final joined graphs (and then associate the primitive elements that make up each joined graph). Figure 66 illustrates a process 6600 of some embodiments for processing a cluster into subsequences. In some embodiments, process 6600 is performed at operation 6450 of process 6400, for each cluster.

As shown, process 6600 receives (at 6605) a cluster of graphs. As noted, in some
embodiments, this cluster is the output of bounds clustering that uses spreads as the first order differences for graphs that are ordered by drawing order. The process then determines (at 6607) whether the cluster is empty (i.e., does not include any graphs). When the cluster is empty, the process ends. Otherwise, the process selects (at 6610) the first graph in the cluster that is not yet in a subsequence. In some embodiments, the cluster is ordered by drawing order, such that the first time through operation 6610 the selected graph is the first graph in the cluster that is drawn in the document that includes the graphs.

[00420] The process then defines (at 6615) a new subsequence that includes the selected graph (at this point, the selected graph is the only graph in the subsequence). The new subsequence has the bounds of the selected graph. In some embodiments, the bounds of the selected graph is the smallest upright bounding box that includes the graph. Other embodiments define the bounds of the graph differently, e.g. using the smallest-area path that completely encloses all of the non-transparent pixels of the graph.

[00421] Next, process 6600 determines (at 6620) whether there are more graphs in the cluster. When there are no more graphs in the cluster, the process proceeds to 6645, which is described below. Otherwise, the process selects (at 6625) the next graph in the cluster. In some embodiments, the next graph in the cluster is the next graph in the drawing order that is in the cluster.

[00422] The process determines (at 6630) whether the bounds of the new graph (i.e., the graph selected at 6625) intersect with the bounds of the current subsequence. As noted above, different embodiments define the bounds of a graph differently. The bounds of a subsequence that includes multiple graphs is described below. When the bounds of the new graph do not intersect the bounds of the current subsequence, process stores (at 6640) the current subsequence (e.g., in a list of subsequences) and proceeds to 6610, which is described above, to begin the next subsequence. The next subsequence begins with the graph recently tested at 6630, because this is the first graph in the cluster that is not yet in a subsequence.

[00423] When the bounds of the new graph (selected at 6625) intersect the bounds of the current subsequence, the process adds (at 6635) the new graph to the subsequence and modifies the bounds of the subsequence to be the intersection of the previous subsequence bounds and the bounds of the newly added graph. The process then proceeds to 6620, described above.
continue attempting to add graphs to the subsequence.

[00424] In some embodiments, the bounds of a subsequence including multiple graphs is the smallest upright bounding box that includes all of the graphs. In other embodiments, the bounds is the union of all of the upright bounding boxes for the graphs in the subsequence (in such embodiments, the bounds of the subsequence will not necessarily be rectangular). In some embodiments that define the bounds of a graph as the smallest-area path including all of the non-transparent pixels of the graph, the bounds might be such a path around all of the graphs in the subsequence or could be the union of such paths for each graph in the subsequence.

[00425] Once all graphs in the cluster have been placed in initial subsequences, the process selects (at 6645) a first subsequence S/. In some embodiments, each subsequence includes graphs that are contiguous in the drawing order and the subsequences are arranged based on the drawing order such that the first subsequence is that with the first graphs in the drawing order.

[00426] The process then determines (at 6650) whether there are more subsequences (i.e., the first time through the process determines whether there is only one subsequence or not). When there are no more subsequences, the process ends. Otherwise, the process selects (at 6655) a next subsequence S2.

[00427] Next, process 6600 determines (at 6660) whether the bounds of S/ and S2 intersect. As described above, the bounds of the subsequences are defined differently in different embodiments (i.e., they are based on upright bounding boxes in some embodiments, paths around the non-transparent pixels in other embodiments, etc.). When the bounds of S/ and S2 do not intersect, the process defines (at 6665) S2 to be S7 and proceeds to 6650 to test the next subsequence against the original S2.

[00428] When the bounds do intersect, the process merges (at 6670) the two subsequences and proceeds to 6645 to select the first subsequence as S/. Some embodiments return to the first subsequence and do not finish processing until a set of subsequences that cannot be merged in any way is run through from the beginning. Other embodiments save processing time, however, by selecting the subsequence prior to the recently merged subsequence as S/ upon returning to 6645 and proceeding from that point rather than starting over at the first subsequence.
Once the clusters have been processed into subsequences, the subsequences can be tested against constraints such as the density constraints described above. Some embodiments require a particular minimum number of graphs in a subsequence for the graphs to be associated in a joined graph (e.g., two, five, etc.). Some embodiments required that the spread (calculated as described above) be less than a particular number (e.g., 0.4, 0.5, etc.).

C. Software Architecture

In some embodiments, the graph joining processes described above are implemented as software running on a particular machine, such as a computer, a media player, a cell phone (e.g., an iPhone®), or other handheld or resource-limited devices (or stored in a computer readable medium). Figure 67 conceptually illustrates a graph joining application 6700 of some embodiments for identifying graphs that should be joined and associating the graphs as one graphic. In some embodiments, the application is a stand-alone application or is integrated into another application (e.g., a document reconstruction application), while in other embodiments the application might be implemented within an operating system.

Figure 67 illustrates a graph joiner 6705, a bounds clustering module 6710, and a spread calculator 6715, as well as document content 6725. Graph joiner module receives information from the document content 6725. In some embodiments, the information is information about the location of each graph and the drawing order of the graphs.

The graph joiner 6705 passes information (e.g., locations of graphs and the position of the graphs in the drawing order) to the spread calculator 6715. The spread calculator 6715 of some embodiments calculates the spread for each successive pair of graphs, and passes this information to bounds clustering module 6710.

Bounds clustering module 6710 receives information from the graph joiner 6705 and the spread calculator 6715 (e.g., an array of spreads to be treated as first order differences) and performs bounds clustering on the received information. The results of the bounds clustering arc passed back to the graph joiner. In some embodiments, the graph joiner 6705 performs further processing of the clusters received from the bounds clustering module to identify whether particular clusters of graphs should be associated as single graphs, and returns the associations to the document content 6725.
[00434] In some embodiments, the results of the processes performed by the above-described modules or other modules are stored in an electronic storage (e.g., as part of a document object model). The document object model can then be used for displaying the document on an electronic display device (e.g., a handheld device, computer screen, etc.) such that a user can review and/or interact with the document (e.g., via touchscreen, cursor control device, etc.).

VII. PROFILES FOR CONTROLLING PROCESSING

[00435] The interpretation of visual information as structural elements in a document can depend on the type of content that is in the document being processed. Semantic reconstruction (as well as other content processing techniques) can therefore be improved by tailoring the analysis to the type of content. Some embodiments provide methods for identifying a type of content to be processed, and tailoring the processing of the content to the identified type of content. Some embodiments apply the tailoring of content processing based on identification of the type of content to document reconstruction as described above in Sections U-VI.

[00436] Some embodiments use profiles to identify the type of content to be processed. A profile includes a logical proposition of the form \( P_1 \land P_2 \land P_3 \land \ldots \land P_s \), such that the profile matches particular content if and only if all logical predicates \( P_v \) are true for the particular content. In some embodiments, the set of profiles is hierarchical. A hierarchical set of profiles has profiles for different levels of hierarchy that are reached at different stages in the processing.

In some embodiments, instructions from a lower level profile can override instructions from a higher level profile.

[00437] Figure 68 conceptually illustrates a process 6800 of some embodiments that uses profiles for tailoring content processing to content type. As shown, the process receives (at 6805) content. The content is a document to be reconstructed in some embodiments. In other embodiments, the content is a document, but the processing to be performed on the document is analysis other than reconstruction. The content can also be other types of content (e.g., audio and/or video content) in some embodiments. For instance, some embodiments could analyze video content to break the video content into a hierarchy of scenes, acts, etc., and modify later stages of analysis based on the results of earlier stages.

95
[00438] Next, the process identifies (at 6810) the type of content received based on profiles for different content types. In some embodiments, a profile includes a logical proposition comprised of one or more logical predicates that matches content if and only if all of the logical predicates are true for that content. In some embodiments, a profile also specifies how to perform content processing if the profile matches the content. In the case of document processing, a profile includes a collection of attributes that apply to a collection of visual elements in some embodiments. For example, a simplified profile for a document intended to be printed on a #10 size envelope might be (\# of pages = 1 \land \# of paragraphs \leq 2 \land width = 9.5" \land height = 4.125").

[00439] **Figure 69** illustrates a document 6900. In identifying the document type, some embodiments will test multiple profiles until arriving at a profile all of whose logical predicates are true. In the case of document 6900, if width x is 9.5" and height y is 4.125", then the document 6900 would be identified as a #10 envelope, as it only has one page with two paragraphs 6905 and 6910.

[00440] Once the type of content is identified, process 6800 performs (at 6815) content processing tailored to the identified content type. The process then ends. In some embodiments, tailoring the processing to the identified content type includes adding and removing processes, modifying how the processes are performed, and modifying the order in which the processes are performed. For example, in the case of document reconstruction, when it is determined that a document is a single page of text lines with no columns, headers, or footers, the document reconstruction processes can be tailored appropriately. Some embodiments will identify text lines, identify the words in the text lines, and group the text lines into paragraphs, but will not perform other processes (e.g., process 4200 for identifying columns and layouts, or table identification processes) because it is recognized based on the document type that these are not required.

A. **Hierarchical Profiling**

[00441] Some embodiments use a hierarchical set of profiles to tailor content processing to content type. **Figure 70** illustrates a hierarchical set of profiles 7000 of some embodiments for document reconstruction. The hierarchical set of profiles 7000 includes two document profiles 7001 and 7002, three section profiles 7005-7007, eight page profiles 7010-7017, and four zone profiles 7020-7023.
As shown, each document profile has its own profile tree. In some embodiments, once a document profile is identified, then only the section profiles in its hierarchical tree will be tested. For instance, when a document is identified as Document Profile A 7001, then only Section Profile A 7005 and Section Profile B 7006 will be tested. Similarly, once a section profile (or any other lower level profile) is identified, then only profiles in that particular profile’s tree will be tested.

In some embodiments, some profiles are shared between multiple profiles of a higher level of hierarchy. For instance, Page Profile C 7012 is shared between all three section profiles 7005-7007. Thus, whether a section of a document is identified as Section Profile A 7005, Section Profile B 7006, or Section Profile C 7007, Page Profile C 7012 will be tested as one of the possible page profiles for each page in the section.

Figure 71 conceptually illustrates a process 7100 of some embodiments that matches profiles against content and dynamically configures content processing depending on the matched profile. As shown, the process receives (at 7105) content to be processed. In some embodiments, the content is a document to be reconstructed according to processes described above in Sections M-VI. In other embodiments, the content is audio and/or video content, or other types of content to be analyzed.

Next, the process selects (at 7110) a profile for the top level in the hierarchy. In some embodiments, selecting a particular profile includes evaluating the logical propositions in the particular profile for the content and identifying that the profile matches the content. In some embodiments some amount of processing must first be performed in order to select a profile from the top level in the hierarchy. In some embodiments, only one profile at a time can be selected at the top level of hierarchy, though multiple profiles may be tested before finding a matching profile. For instance, referring to the hierarchical set of profiles 7000 in Figure 70, a document can match either Document Profile A 7001 or Document Profile B 7002, but once one matches then the matching profile is selected.

After selecting the profile from the hierarchy, the process applies (at 7115) processing for the selected profiles. For instance, referring to document reconstruction, when the content is a simple one-page document, one set of reconstruction methods is applied. On the other hand, when the document is a many-page book with illustrations a different set of
reconstruction methods are applied. Some embodiments do not apply all methods at once, but instead apply one method at a time, in an order specified by the profile, before determining whether a new profile at a lower level can be selected.

[00447] Next, the process determines (at 7120) whether processing is far enough along to proceed to the next level of hierarchy. In some embodiments, this involves determining whether enough of the processing methods have completed and enough information is made available about the next level of hierarchy that the profiles at the next level can be tested for a match. When processing is not far enough along, the process proceeds to 7115 to continue applying processing as described above.

[00448] Once processing is far enough along for the next level of hierarchy, the process selects (at 7125) a profile for the next level of hierarchy. In some embodiments, for each content entity (e.g., section, page, zone, etc.) at a particular level of hierarchy, only one profile can be selected at a time for its particular level. For instance, referring to the hierarchical set of profiles 7000 in Figure 70, a section in a document that matches Document Profile A 7001 can match either Section Profile A 7005 or Section Profile B 7006, but not both. Furthermore, in some embodiments, the selection of a profile for a content entity is restricted based on the selection of profiles for content entities above. For instance, in the above example, because the document matched Document Profile A 7001, a particular section will not be tested against Section Profile C 7007 and therefore cannot match Section Profile C 7007.

[00449] The process next modifies (at 7130) the processing methods from previous levels of the hierarchy based on newly selected profiles. In some embodiments, any lower level profile can override any already-defined processing method from a higher level profile. Modifications can include removing processing methods, modifying the manner in which certain methods are performed, adding methods, changing the order in which methods are performed, etc. For instance, when processes (e.g., process 3800 described above in Section IV) have already been used to split lines in a document, then a determination as to whether there is adequate evidence for columns can determine whether a page is classified as complex or simple. When the page is simple (i.e., minimal evidence for columns), the line-splitting method will be modified in some embodiments to insert tabs in the gaps in the lines rather than actually splitting the lines.

Similarly, when the page is a simple one with no zone border graphics, zone analysis methods
are removed in some embodiments.

[00450] As an example of adding methods, a profile identifying a particular document as a newspaper might add methods to search for text at the end of a layout that specifics that an article continues on a particular page with a particular title in some embodiments. Such a method would then search the specified page for the title in order to define a link between the two layouts (or zones) in the document to indicate that the two layouts (or zones) form one article.

[00451] Next, with the processing methods modified, process 7100 applies (at 7135) the processing methods for the selected profiles. In some embodiments, this includes methods for profiles at a higher level of the hierarchy than the most recently selected profiles. For instance, when a particular method is specified in a document profile and a selected section profile does not modify the particular method or whether the particular method is performed, the particular method will be performed (if it has not been performed already) according to the document profile. Some embodiments do not apply all methods at once, but instead apply one method at a time, in an order specified by the profile, before determining whether a new profile at a lower level can be selected.

[00452] Next, the process determines (at 7140) whether there are any lower levels in the hierarchy of profiles to test. When there are no more lower levels in the hierarchy, the process proceeds to 7150 which is described below. Otherwise, the process determines (at 7145) whether processing is far enough along to proceed to the next level of hierarchy. In some embodiments, this involves determining whether enough of the processing methods have completed and enough information is made available about the next level of hierarchy that the profiles at the next level can be tested for a match. When processing is not far enough along, the process proceeds to 7135 to continue applying processing as described above. Once processing is far enough along for the next level of hierarchy, the process proceeds to 7125 to select one or more profiles for the next level of hierarchy, as described above.

[00453] When the process determines (at 7140) that there are no more lower levels of hierarchy, the process determines (at 7150) whether the content processing is complete. In some embodiments, content processing is complete when all of the processing methods have been applied for the lowest level profile. When processing is complete, the process ends. Otherwise, the process proceeds to 7135, which is described above, to continue processing the content.
B. Using Hierarchical Profiling for Implementing Semantic Reconstruction Methods

[00454] Some embodiments utilize hierarchical profiling to more efficiently and accurately implement the semantic reconstruction methods described above in Sections H-VI. In some embodiments, hierarchical profiles are used to add, remove, modify, or re-perform the methods. Some embodiments include a default profile that, in the absence of any profiles matching, implements all possible reconstruction methods. When a profile lower in the hierarchy is selected, however, the selected profile can override the default profile regarding whether to perform a particular method and/or how to implement a particular method.

[00455] In some embodiments, profile matching occurs at different stages in the processing. As such, the set of profiles activated at any given time can change throughout processing, and the inheritance of methods has to be reassessed after each new profile is activated. Because a newly activated profile can reorder, add, or remove processing methods, some embodiments also change the effect of inheritance during the content processing.

[00456] Figure 72 conceptually illustrates a process 7200 of some embodiments for performing document reconstruction using hierarchical profiling. As shown, the process receives (at 7205) a document. In some embodiments, the document is a vector graphics document (e.g., a PDF document) that is to be reconstructed using semantic reconstruction techniques.

[00457] Next, process 7200 identifies (at 7210) a profile for the document. In some embodiments, initially a default profile is assigned that implements all possible reconstruction methods. Some embodiments, however, can identify a document profile based on the inherent properties of the document (e.g., number of pages, size of pages, etc.) without applying any reconstruction methods.

[00458] Once the initial profile is identified, the process identifies (at 7215) reconstruction methods associated with the identified profile. When the initial profile is simply the default profile, all possible methods will be identified. However, when the profile is an identified document profile or a profile at a lower level in the hierarchy, in some cases only some of the reconstruction methods will be identified. For example, some profiles do not explicitly define an implementation for performing a particular process, but also do not specify not to perform the
process.

[00459] The process then selects (at 7220) one of the identified methods. In some embodiments, the method selected is based on a specified order for performing the reconstruction methods. For instance, the default profile of some embodiments specifies a default order for implementing the reconstruction methods. The default order, e.g., may specify performing zone analysis methods first, then identifying text lines, then performing guide and gutter analysis, etc.

[00460] In some embodiments, identifying methods include identifying methods that are excluded by the current profile. For instance, reconstruction processes might have already identified potential zone borders for a page. In some embodiments, when there is at least a particular density of zone borders, and text patterns match appropriately, the page is identified as being a street map. Accordingly, the remainder of zone analysis (e.g., identifying zones) will not be performed, because this would break the page into an unreasonable number of small zones that were not intended by the author of the document to be separate areas of a page.

[00461] Similarly, a document profile specifying that a document is a word processing document includes a method for column identification in some embodiments. However, when a particular page in the document has embedded content, the profile for the particular page could override the column identification step with instructions not to perform that method.

[00462] Once a method is selected, process 7200 selects an implementation for the method. In some embodiments, active (i.e., selected) profiles can implement a method, or can inherit an implementation of the method from an active profile higher in the hierarchy. The process determines (at 7225) whether the selected method is explicitly defined for the current (i.e., lowest active level) profile. When the selected method is explicitly defined, the process selects (at 7230) the implementation of the method for the current profile. The process then proceeds to 7250, which is described below.

[00463] Otherwise, the process determines (at 7235) whether the selected method is defined for an active profile at a higher level of the hierarchy. When the process is defined for an active profile higher in the hierarchy, the process selects (at 7240) the implementation of the method from the higher level active profile. The process then proceeds to 7250 which is
described below. Otherwise, the method selects (at 7245) the default implementation of the method.

[00464] **Figure 73** illustrates how some embodiments inherit the lowest level active profile that implements a particular method. The figure shows a hierarchy 7300 of document reconstruction profiles. The hierarchy includes a set 7305 of default implementations for all methods, sets 7310 of implementations for various document profiles, sets 7315 of implementations for various section profiles, sets 7320 of implementations for various page profiles, and sets 7325 of implementations for various zone profiles. As shown by the dashed lines, Zone Profile 1 7330 is the current (lowest-lvcl) active profile, while Page Profile 1 7335, Section Profile X 7340, and Document Profile 2 7345 are also active.

[00465] As Zone Profile 1 7330 is the current active profile, an attempt to implement a particular method (e.g., guide identification) looks to the set of implementations for Zone Profile 1 7330 to determine whether it explicitly defines an implementation for the particular method. When Zone Profile 1 7330 docs explicitly define an implementation, then this implementation is selected. Otherwise, the active Page Profile 1 7335 is examined. Active profiles are examined moving up the hierarchy until either an implementation of the particular method is defined or the set 7305 of default implementations for all methods are reached. When the lowest level active profile that implements a method is identified, this implementation is inherited by the current active profile. In no active profiles implement a method, then the current active profile inherits the implementation from the default profile.

[00466] After process 7200 selects an implementation for a method, the process performs (at 7250) reconstruction using the selected implementation. In some embodiments, the reconstruction performed includes one or more of the processes described above in Sections II-VI.

[00467] Next, the process determines (at 7255) whether a new profile is identified. Some embodiments, after each method is performed, send all of the reconstruction data available to a profile matching engine. The profile matching engine of some embodiments determines, based on the information received, whether there is enough information to test particular profiles (e.g., profiles at the next level of hierarchy below the current active profile). If there is enough information, the profile matching engine tests the information...
predicates in the profiles. Some embodiments order the profiles so that those most likely to
match or requiring the least amount of work will be tested first. Profile matching is described in
further detail below in subsection C.

[00468] When a new profile is not identified, the process determines (at 7265) whether
reconstruction of the document is complete. In some embodiments, reconstruction is complete
when all methods specified by the set of active profiles (and not removed by a lower-level active
profile) are completed. When reconstruction is complete, the process ends. Otherwise, the
process proceeds to 7220 to select the next identified method as described above.

[00469] When a new profile is identified, process 7200 removes (at 7260) any
reconstruction results specified by the new profile. Specifically, in addition to specifying
particular methods that should be performed or not performed, profiles in some embodiments
specify that results of particular methods should be undone. For example, even after zone
analysis methods have been performed, a profile might specify that the zone information
(borders, intersections, zones, etc.) should be removed from the results and that zone analysis
should be performed again after several other methods are performed. The process then proceeds
to 7215, which is described above, and proceeds until reconstruction is complete.

C. Profile Matching

[00470] As noted above, in some embodiments, a profile is a set of logical predicates that
must all be true for the profile. Some embodiments use a profile matching engine that receives
content processing results, determines whether enough information is present to test the next
level of profiles for the current active profile, and then tests the content processing results
against the profiles. Some embodiments specify a particular order for testing profiles based on
the order in which results are likely to arrive as well as on which profiles are most likely to
match.

[00471] Figures 74-77 illustrate an example page profile matching process of some
embodiments for four different pages. Figure 74 illustrates four page profiles: Simple Page
7401, Standard Page 7402, Rich Page 7403, and Layout Page 7404. Each of the profiles includes
at least one logical predicate: specifically profile 7401 includes three predicates 741 1-7413,
profile 7402 includes four predicates 7421-7424, profile 7403 includes two predicates 7431 and
7432, and profile 7404 includes only one predicate.

[00472] Figure 74 also illustrates page 7400 that is tested against the page profiles in the order shown. Page 7400 includes only a graphic object 7450 and text lines 7460. The body of page 7400 has only one child (the page borders, which has no children itself), and the profile for the zone is simple because there is only one column of text in one layout. In some embodiments, once a profile is matched, the others are not tested. The dashed lines in Figure 74 illustrate the path through the predicates used by the profile matching engine in order to determine a page profile for page 7400. First, the engine determines whether the body of the page has only one child (predicate 7411). Because this predicate is evaluated as true, the engine determines whether the child has no zone children (predicate 7412, also true). Finally, the engine determines whether the zone profile is simple (i.e., there is only one column of text in one layout) (predicate 7413, also true). As all three predicates are true for page 7400, the profile matching engine determines that page 7400 is a simple page. Accordingly, certain methods (e.g., column identification) can be removed for page 7400.

[00473] Figure 75 illustrates a second page 7500 that is tested against the page profiles 7401-7404. Page 7500 includes a graphic object 7540, header lines 7545, and two columns of text lines 7550. The body of page 7500 has only one child (the page borders, which has no children itself), and the text in the zone is a header and then two columns of text. The dashed lines in Figure 75 illustrate the path through the predicates used by the profile matching engine in order to determine a page profile for page 7500. First, the engine determines whether the body of the page has only one child (predicate 7411). Because this predicate is evaluated as true, the engine determines whether the child has no zone children (predicate 7412, also true). The engine then determines whether the zone profile is simple (predicate 7413, false). Because there is a header and two columns, the zone profile is not simple, and thus predicate 7413 is false and the matching engine checks the next profile. Predicates 7421-7424 are all true for page 7500, and therefore the profile matching engine determines that page 7500 is a standard page. While these are also all true for page 7400 in Figure 74, because the Simple Page profile 7401 is evaluated first (and matches), the Standard Page profile 7402 was never checked for page 7400.

[00474] Figure 76 illustrates a third page 7600 that is tested against the page profiles 7401-7404. Page 7600 includes a main zone with several children including a header zone with
text, a sidebar, a main layout around a centered call-out, and a bottom zone with text. The dashed lines in Figure 76 illustrate the path through the predicates used by the profile matching engine in order to determine a page profile for page 7600. First, the engine determines whether the body of the page has only one child (predicate 7411). Because this predicate is evaluated as true, the engine determines whether the child has no zone children (predicate 7412, false). Because the primary zone has numerous children, predicate 7412 is false and the matching engine checks the next profile. Predicate 7422 is false for the same reason, so the matching engine moves to the third profile, and predicates 7431 and 7432 are evaluated as true as the number of layouts is small and the page has a main flow layout (the layout that wraps around the concentric squares).

Accordingly, page 7500 is classified as a rich page.

Figure 77 illustrates a fourth page 7700 that is tested against the page profiles 7401-7404. Page 7700 includes a large zone with graphics and an island with text, as well as a smaller zone with text. The dashed lines in Figure 77 illustrate the path through the predicates used by the profile matching engine in order to determine a page profile for page 7400. First, the engine determines whether the body of the page has only one child (predicate 7411). Because this predicate is evaluated as false, the matching engine checks the next profile (which fails for the same reason). On the third profile, the predicate 7432 is false because the page 7700 does not have a main flow layout. Predicate 7441 is always true, and thus the page 7700 is classified as a layout page. Predicate 7441 is always true because once a page has failed to match any previous profiles, it is automatically classified as a layout page. Once the profile is matched, reconstruction methods can be performed, removed, etc. according to the instructions in the profile.

VIII. IDENTIFICATION AND SELECTION OF REGIONS OF INTEREST, AND NAVIGATION AND DISPLAY OF DOCUMENTS

Document viewing applications such as an e-book reader will often need to know how to best display a document and navigate within a document. This is especially important on small-screen devices that cannot legibly display entire pages of documents at once. For instance, in some cases, a document viewer should be able to recognize that an entry in the table of contents links to a particular section of the document, or that two sections of a document are related (e.g., that a call-out is part of a particular article, or that one column flows into the next).
Some embodiments of the invention use information gained from document reconstruction (i.e., paragraph and column information, table information, etc.) for display and navigation of a document. Specifically, some embodiments adapt display and navigation of semantically reconstructed documents for display and navigation on small-screen devices (e.g., media players, cell phones, etc.).

A. Identification and Selection of Regions of Interest

Some embodiments provide methods for identifying and selecting a region of interest in a semantically reconstructed document, and then modifying the display of the document based on the selection of the region of interest. Figure 78 conceptually illustrates a process 7800 for displaying a document based on an identification of a position of interest in some embodiments. Process 7800 will be described in conjunction with Figure 79. Figure 79 illustrates a sequence 7900 (7905 - 7920) on a small-screen device in which a position of interest is selected and the display is modified in accordance with some embodiments of the invention.

As shown in Figure 78, process 7800 receives (at 7805) an indication of a position of interest in a semantically reconstructed document. In some embodiments, the semantically reconstructed document includes a document object model that is the result of document reconstruction as described above in Sections H-VI. A document object model in some embodiments includes a zone graph as described above in Section II, after it has been populated with content information throughout the document reconstruction process. The document object model of some embodiments also indicates the reading order of the content (e.g., of the columns, paragraphs, images, etc.).

Some embodiments receive an indication of a position of interest as a selection of a point on a display. For instance, a selection can be made with a cursor control device (e.g., a mouse, louchpad, etc.). A position of interest can also be indicated on a touchscreen device by a user tapping the screen (e.g., a single-tap, double-tap, pinching motion, etc.). Referring to Figure 79, sequence 7900 shows (at 7905) a user 7925 selecting a point on the display 7930, which is displaying a portion of a document. Position of interest 7935 is illustrated (at 7910) at the point in the display 7930 where the user touched the screen to make the selection.

Next, process 7800 identifies (at 7810) a region of interest in the semantically

[00477] Some embodiments of the invention use information gained from document reconstruction (i.e., paragraph and column information, table information, etc.) for display and navigation of a document. Specifically, some embodiments adapt display and navigation of semantically reconstructed documents for display and navigation on small-screen devices (e.g., media players, cell phones, etc.).

A. Identification and Selection of Regions of Interest

Some embodiments provide methods for identifying and selecting a region of interest in a semantically reconstructed document, and then modifying the display of the document based on the selection of the region of interest. Figure 78 conceptually illustrates a process 7800 for displaying a document based on an identification of a position of interest in some embodiments. Process 7800 will be described in conjunction with Figure 79. Figure 79 illustrates a sequence 7900 (7905 - 7920) on a small-screen device in which a position of interest is selected and the display is modified in accordance with some embodiments of the invention.

As shown in Figure 78, process 7800 receives (at 7805) an indication of a position of interest in a semantically reconstructed document. In some embodiments, the semantically reconstructed document includes a document object model that is the result of document reconstruction as described above in Sections H-VI. A document object model in some embodiments includes a zone graph as described above in Section II, after it has been populated with content information throughout the document reconstruction process. The document object model of some embodiments also indicates the reading order of the content (e.g., of the columns, paragraphs, images, etc.).

Some embodiments receive an indication of a position of interest as a selection of a point on a display. For instance, a selection can be made with a cursor control device (e.g., a mouse, louchpad, etc.). A position of interest can also be indicated on a touchscreen device by a user tapping the screen (e.g., a single-tap, double-tap, pinching motion, etc.). Referring to Figure 79, sequence 7900 shows (at 7905) a user 7925 selecting a point on the display 7930, which is displaying a portion of a document. Position of interest 7935 is illustrated (at 7910) at the point in the display 7930 where the user touched the screen to make the selection.
reconstructed document. Sequence 7900 shows (at 7915) that the paragraph 7940 has been identified as the region of interest based on selection of position of interest 7935. Some embodiments identify a paragraph as the region of interest when the position of interest is within the bounding box of the paragraph, as is the case with position of interest 7935.

[00482] After identifying the region of interest, process 7800 applies (at 7815) required transformations to place the region of interest in the viewing area of the display device. In some embodiments, the transformations include a combination of rotations, zooms, and translations, as described below with respect to processes 8000 and 8100. Next, the process draws (at 7820) the document based on the applied transformations. The process then ends. Sequence 7900 shows (at 7920) that paragraph 7940 has been zoomed in and centered both vertically and horizontally, according to specified transformations. In some embodiments, the process displays the original (i.e., unstructured) document, but uses the knowledge of the positions of the structural elements (e.g., from the structured document) to pan and zoom the document.

[00483] Figure 80 conceptually illustrates a detailed process 8000 of some embodiments for identifying a region of interest and applying transformations to draw the region of interest on a display device. As shown, process 8000 receives (at 8005) a position of interest. In some embodiments, the position of interest is used based on a selection by a user (e.g., a selection with a cursor control device, or through a touchscreen).

[00484] Based on the position of interest, the process then determines (at 8010) a selected object in a semantically reconstructed document based on a received position of interest. In some embodiments, a selected object can be a character, word, text line, image, etc. In some embodiments, the semantically reconstructed document includes a document object model (DOM) that is the result of document reconstruction as described above in Sections H-VI. A document object model in some embodiments includes a zone graph as described above in Section II, after it has been populated with content information throughout the document reconstruction process.

[00485] The process next moves up (at 8015) the document hierarchy until either a paragraph or graphic object is reached. In some embodiments, the document hierarchy is the zone graph populated with content information (i.e., the DOM). When the selected object is a word, in some embodiments the process moves up the hierarchy from the word to the text line
including the word to the paragraph including the text line.

[00486] The process then determines (at 8020) whether the identified object is a paragraph. When the identified object is not a paragraph, it is a graphic object. In some embodiments, the graphic object can be any of a shape, image, or joined graph (i.e., a compound graphic object). When the identified object is a graphic object, the process defines (at 8025) a rectangle of interest (ROI) as the upright bounding box of the identified object. The process then zooms (at 8030) such that the width and height of the ROI are no larger than the width and height of the display area. The process then proceeds to 8060, which is described below.

[00487] When the identified object is a paragraph, the process defines (at 8035) a rectangle with the width of the parent column of the paragraph and the height of the paragraph. The process then applies (at 8040) any rotation for the paragraph to the rectangle and defines the ROI as the upright bounding box of the rotated rectangle. A paragraph is rotated if it is in a rotation group in the document object model in some embodiments. Some embodiments define a rotation group as described above in Section II.

[00488] The process then determines (at 8045) whether the paragraph baselines (prior to rotation) are steeper than the diagonal of the display area. For instance, when the display area is a square, the determination is whether the baselines are steeper or less steep than 45 degrees. When the paragraph baselines are steeper than the diagonal of the display area, the process zooms (at 8050) such that the ROI is no larger than the height of the display area. The process then proceeds to 8060 which is described below. On the other hand, when the paragraph baselines are not steeper than the diagonal of the display area, the process zooms (at 8055) such that the ROI is no larger than the width of the display area.

[00489] Next, process 8000 determines (at 8060) whether the ROI (as defined in the operations above) fits in the display area horizontally. When the ROI fits horizontally, the process centers (at 8065) the ROI horizontally. The process then proceeds to 8075 which is described below. Otherwise, the process centers (at 8070) the position of interest horizontally. The process then determines (at 8075) whether the ROI fits in the display area vertically. When the ROI fits vertically, the process centers (at 8080) the ROI vertically. The process then proceeds to 8087 which is described below. Otherwise, the process centers (at 8085) the position of interest vertically.
Next, the process determines (at 8087) whether either the bottom or top edge of the page that includes the position of interest is in the display area. When so, the process moves (at 8090) the visible page edge to the corresponding (i.e., top or bottom) edge of the display area so that only one page is displayed and the entire display area is occupied vertically. The process then determines (at 8092) whether either the right or left edge of the page that includes the position of interest is in the display area. When so, the process moves (at 8095) the visible page edge to the corresponding (i.e., right or left) edge of the display area so that only one page is displayed and the entire display area is occupied vertically. Finally, the process redraws (at 8197) the display area with all of the above transformations applied. The process then ends.

Some embodiments use other combinations of transformations (e.g., zooms, translations, rotations, etc.) to display a selected region of interest. For instance, while process 8000 determines how to zoom differently for a region of interest that is a paragraph as compared to a region of interest that is a graphic object, other embodiments define a region of interest such that the zoom is the same for all regions of interest. For instance, process 8100, described below, zooms on a region of interest in this manner.

Figure 81 conceptually illustrates a detailed process 8100 of some embodiments for identifying a region of interest and applying transformations to draw the region of interest on a display device. Process 8100 is similar to process 8000, described above. In some embodiments, a user can select whether they prefer the zoom transformations in process 8000 or process 8100. Other embodiments implement one of the two processes. As shown, process 8100 receives (at 8105) a position of interest. In some embodiments, the position of interest is used based on a selection by a user (e.g., a selection with a cursor control device, or through a touchscreen).

Based on the position of interest, the process then determines (at 8110) a selected object in a semantically reconstructed document based on a received position of interest. In some embodiments, a selected object can be a character, word, text line, image, etc. In some embodiments, the semantically reconstructed document includes a document object model that is the result of document reconstruction as described above in Sections H-VI. A document object model in some embodiments includes a zone graph as described above in Section II, after it has been populated with content information throughout the document reconstruction process.
[00494] Process 8.115 next moves (at 8.115) up the document hierarchy until either a paragraph or graphic object is reached. In some embodiments, the document hierarchy is the zone graph populated with content information. If the selected object is a word, then in some embodiments the process moves up the hierarchy from the word to the text line including the word to the paragraph including the text line.

[00495] The process then determines (at 8.120) whether the identified object is a paragraph. If the identified object is not a paragraph, then it is a graphic object. In some embodiments, the graphic object can be any of a shape, image, or joined graph (i.e., a compound graphic object). When the identified object is a graphic object, the process defines (at 8.125) a rectangle of interest (ROI) as the upright bounding box of the identified object, then proceeds to 8.150, which is described below.

[00496] When the identified object is a paragraph, the process defines (at 8.135) a rectangle with the width of the parent column of the paragraph and the height of the paragraph. The process then applies (at 8.140) any rotation for the paragraph to the rectangle. A paragraph is rotated if it is in a rotation group in the document object model in some embodiments. Some embodiments define a rotation group as described above in Section II.

[00497] The process then defines (at 8.145) the ROI as the upright bounding box that includes a first particular number of lines above the position of interest and a second particular number of lines below the position of interest. This definition is based on the assumption that a user is interested in the position of interest and would want to see a certain number of lines of text below and above that position of interest.

[00498] With the ROI defined, process 8.100 then zooms (at 8.150) such that the width and height of the ROI are no larger than the width and height of the viewing area.

[00499] Next, process 8.100 determines (at 8.160) whether the ROI (as defined in the operations above) fits in the display area horizontally. When the ROI fits horizontally, the process centers (at 8.165) the ROI horizontally. Otherwise, the process centers (at 8.170) the position of interest horizontally. The process then determines (at 8.175) whether the ROI fits in the display area vertically. When the ROI fits vertically, the process centers (at 8.180) the ROI vertically. Otherwise, the process centers (at 8.185) the position of interest vertically.
After centering the position and/or region of interest, the process determines (at 8 187) whether either the bottom or top edge of the page that includes the position of interest is in the display area. If so, the process moves (at 8190) the visible page edge to the corresponding (i.e., top or bottom) edge of the display area so that only one page is displayed and the entire display area is occupied vertically. The process then determines (at 8192) whether either the right or left edge of the page that includes the position of interest is in the display area. If so, the process moves (at 8195) the visible page edge to the corresponding (i.e., right or left) edge of the display area so that only one page is displayed and the entire display area is occupied vertically. Finally, the process redraws (at 8197) the display area with all of the above transformations applied, then ends.

B. Display and Navigation of Semantical!] Reconstructed Documents

Some embodiments provide various methods for optimizing display and navigation of semantically reconstructed documents. In some embodiments, the display and navigation is optimized for small-screen devices (e.g., media players, cell phones, etc.).

Figure 82 conceptually illustrates a process 8200 of some embodiments for displaying a semantically reconstructed document. As shown, the process receives (at 8205) a semantically reconstructed document. In some embodiments, the semantically reconstructed document includes a document object model that is the result of document reconstruction as described above in Sections U-VI. A document object model in some embodiments includes a zone graph as described above in Section II, after it has been populated with content information throughout the document reconstruction process.

The process then divides (at 8205) the semantically reconstructed document into sections. In some embodiments, the division is based on information in the document object model for the document. In some embodiments, each section is an ordered sequence of words, associated graphic objects, and other, nested sections. For example, nesting can appear as a boxed example or note, or as cells of a table.

Process 8215 next adapts the viewing of the reconstructed document to accommodate the natural flow through the sections. The process then ends. The adaptations can manifest themselves in multiple ways in some embodiments. Figure 83 illustrates one such
adaptation of some embodiments. **Figure 83** illustrates a device 8305 that is initially displaying a portion of a semantically reconstructed document in two columns 8310 and 8315. In some instances, the document initially had two columns, while in other cases the document has already been adapted for viewing on the device 8305.

Figure 83 illustrates that the device 8305 is turned on its side. The device includes an accelerometer or other such device to recognize that it has been turned on its side. Accordingly, the view rotates 90 degrees so that the text runs horizontally. However, the same two columns 8310 and 8315 are not displayed. Instead, the text from these columns is broken up into three columns 8320-8330 such that all the text is still displayed in the same size, and readability is maintained. This is possible because the document object model, put together by the semantic reconstruction process, indicates that the reading order of the text flows from top to bottom in the first column and then back to the top of the first column.

**Figure 84** conceptually illustrates a process 8400 of some embodiments for navigating between regions in a semantically reconstructed document. As shown, process 8400 displays (at 8405) a region of a semantically reconstructed document. In some embodiments, the semantically reconstructed document includes a document object model that is the result of document reconstruction as described above in Sections H-VI. A document object model in some embodiments includes a zone graph as described above in Section II, after it has been populated with content information throughout the document reconstruction process. The regions of the semantically reconstructed document are defined by the document object model in some embodiments and can include text regions, graphics, etc.

Process 8400 receives (at 8410) input scrolling past the end of the displayed region. The end can be the bottom (scrolling down) or the top (scrolling up) of the region in some embodiments. For instance, when part of a column of text is displayed, the process receives input scrolling up to the top of the column or down to the end of the column.

The process then automatically moves (at 8415) the next region in the flow of the document into the display using the semantically reconstructed hierarchical model of the document (i.e., the document object model). As discussed above, using the document object model, the process can recognize a flow through the text and associated graphics in the document. To continue with the column example, when a πrπr πmlU down past the end of a
column, some embodiments jump to the top of the next column. Some embodiments instead append the text from the top of the second column to the bottom of the first column and continue scrolling as though the text is all one column.

[00509] Figure 85 conceptually illustrates a process 8500 of some embodiments for modifying a layout of a semantically reconstructed document for a small-screen device. As shown, the process receives (at 8505) a semantically reconstructed document with a particular layout. In some embodiments, the semantically reconstructed document includes a document object model that is the result of document reconstruction as described above in Sections H-VI. A document object model in some embodiments includes a zone graph as described above in Section II, after it has been populated with content information throughout the document reconstruction process.

[00510] Figure 86 illustrates a page 8600 of a document as it would be displayed on a large screen or without semantic reconstruction, with a particular two column layout, each column including several text lines. The first column on page 8600 includes a set of words 8605 spanning six full text lines and part of a seventh.

[00511] Process 8500 modifies (at 8510) the layout for viewing on a small-screen device while maintaining the flow of the document by using the hierarchical model of the document (i.e., document object model). The process displays (at 8515) at least a portion of the modified layout on a small-screen device. The process then ends. Figure 86 illustrates a small-screen device 8610 displaying the text 8605 from the first column of page 8600. However, the text has been adapted for ideal viewing on the small-screen device. Rather than display the text very small so that a user has to squint, the text size is increased and the number of words per line is decreased. Scrolling down would enable a user to continue reading the text in order in some embodiments.

[00512] In some embodiments, the document object model for a document includes suggestions regarding how to adaptively display layouts for a variety of situations, display area sizes, and display area aspect ratios. These suggestions can be generated during the document reconstruction process based on the results of the reconstruction, can be set as user preferences, or can be set by the initial author of the document in some embodiments. Some embodiments store the reconstructed document information (e.g., u\V Hnrnpnt nhiorh moHnh with the
document, and can store such user preferences with the reconstructed document. Some embodiments store the document information in XML format, or as metadata of a PDF (or similar format) document.

IX. SELECTION OF TEXT ACROSS LAYOUTS

In addition to display and navigating through documents, the ability to select text may be of great import to a user. Complex pages with multiple different text flows (e.g., multiple articles) which are in turn broken into multiple layouts may create difficulties for an application attempting to intelligently select text in order. When selecting text (e.g., for copying and pasting), it is critical that the ordering of the characters, words, text lines, etc., be preserved. This includes not just selections within a column, but selections that span across multiple columns, layouts, zones, or pages.

Some embodiments provide methods for selecting text within a semantically reconstructed document. In some embodiments, the semantically reconstructed document includes a document object model that is the result of document reconstruction as described above in Sections II-VI. A document object model in some embodiments includes a zone graph as described above in Section II, after it has been populated with content information throughout the document reconstruction process. The document object model of some embodiments includes layout and flow information, such as reading order of columns within a layout, and flow from one layout to the next.

Figure 87 conceptually illustrates a process 8700 of some embodiments for determining a selection of text in a semantically reconstructed document. As shown, process 8700 receives (at 8705) zone and layout information for a document. In some embodiments, zone information includes a zone graph, populated with the content of the document, as described above in Section M. Layout information, in some embodiments, includes columns and layouts as described above in Section IV, as well as the flow of reading through the layouts and between layouts.

The process then displays (at 8710) a portion of the document. In some embodiments, the document is displayed in its original viewing dimensions on a standard screen. Some embodiments, though, display the document in an adaptive manner using one of the
processes described above in Section VIII (e.g., if displaying the document on a small-screen device).

[00517] Process 8700 receives (at 8715) a start point of a selection. The process also receives (at 8720) an end point of the selection. Some embodiments receive start and end points through a cursor control device (e.g., a mouse), a keyboard, or a combination thereof. For example, a selection can be defined by a user clicking on a start point in the text with a mouse, holding the mouse button down, dragging to an end point in the text, and releasing the mouse button. Similar processes can be performed with other cursor control devices. Some embodiments allow other sorts of selections with cursor control devices as well - e.g., double-clicking to select a word, or triple-clicking to select a line of text. A user can also, in some embodiments, use selection keys on a keyboard (e.g., shift and arrow keys) to select text in a semantically reconstructed document.

[00518] In the case of a drag selection, some embodiments define the start point as the point at which the mouse was first clicked and the end point as the current point of the cursor if the mouse button is held down or the release point of the mouse button if it has been released. For double- and triple-clicks, or other such selection mechanisms that select a structural element (e.g., a paragraph, column, etc.), some embodiments define the start point as the bottom left and the end point as the top right of the bounding box of the structural element. In some embodiments, the bounding box of the structural element is the bounding box of the union of its typographic bounds. In some embodiments, the typographic bounds for a character extend from its anchor point on the left to the anchor point plus the width (possibly adjusted by kerning with the following character) on the right and from the ascent above the anchor to the descent below the anchor.

[00519] The process then determines (at 8725) the selected section of text using the start point, end point, and layout and flow properties of the document. The process then ends. Various embodiments provide different detailed processes, some of which are described below, for determining exactly which text is selected from which layouts in a semantically reconstructed document given a start and end point in the document.

[00520] **Figure 88** illustrates a sequence 8800 of a selection of text on a page 8801 of a semantically reconstructed document. The page 8801 includes two main columns 880S and 8810
and a small call-out 8815 (that is a different layout) in the center of the page. The sequence 8800 also shows start point 8820 and end point 8825 for a selection. The selection 8830 of text that results from these start and end points runs from the start to the bottom of the first column, then from the top of the second column to the end point. Text in the center call-out 8815 is not selected in the depicted embodiment. One of ordinary skill in the art will recognize that variations in the selection processes are possible such that the center call-out 8815 will be included in the selection 8830.

[00521] Figure 89 conceptually illustrates a detailed process 8900 of some embodiments for defining a selection of text in a semantically reconstructed document. As shown, process 8900 receives (at 8905) zone and layout information for a document. In some embodiments, zone information includes a zone graph, populated with the content of the document, as described above in Section II. Layout information, in some embodiments, includes columns and layouts as described above in Section IV, as well as the flow of reading through the layouts and between layouts.

[00522] Next, the process assigns (at 8910) a reading order to each column in the document. Some embodiments start at a reading order of zero on each page (i.e., the first column that would be read is assigned reading order zero). Some embodiments impose the requirement that within a particular layout, the reading order assigned to each column must be consistent with the order of the columns in the layout and no column outside a particular layout can have a reading order between those of the starting and ending columns of the particular layout. As for which layout comes first in the reading order, that decision is arbitrary in some embodiments. Other embodiments discern a reading order from layout to layout based on the design of the page.

[00523] The process then displays (at 8915) a portion of the document. In some embodiments, the document is displayed in its original viewing dimensions on a standard screen. Some embodiments, though, display the document in an adaptive manner using one of the processes described above in Section VIII (e.g., when displaying the document on a small-screen device).

[00524] The process then receives (at 8920) start and end points for a text selection. Some embodiments receive start and end points through a cursor or mouse.
keyboard, or a combination thereof. For instance, a selection can be defined by a user clicking on
a start point in the text with a mouse, holding the mouse button down, dragging to an end point
in the text, and releasing the mouse button. Similar processes can be performed with other cursor
control devices. Some embodiments allow other sorts of selections with cursor control devices as
well - e.g., double-clicking to select a word, or triple-clicking to select a line of text. A user can
also, in some embodiments, use selection keys on a keyboard (e.g., shift and arrow keys) to
select text in a semantically reconstructed document.

[00525] Next, the process determines (at 8925) whether the start and end points are in the
same layout. When the start and end points are not in the same layout, the process determines (at
8930) a new end point in the same layout as the start point. Some embodiments move the end
point to the same layout as the start point on the assumption that the user probably accidentally
drifted the selection device into the second layout, and because selections can be determined
more reliably within a single layout as opposed to across layouts. Some embodiments define a
line from the start to the end point, and the location where the line leaves the layout of the start
point is defined as the new end point. Other embodiments translate the end point horizontally or
vertically into the layout of the start point.

[00526] The process then determines (at 8935) whether the start point is before the end
point in reading order. In some embodiments, when the start and end point are in the same
column, reading order is determined such that the higher (and if tied, left-most) of the two points
is the earlier point in the reading order. When the start point is after the end point in the reading
order, the process switches (at 8940) the start and end point of the selection, such that the start
point is always earlier in the reading order than the end point.

[00527] Next, process 8900 determines (at 8945) whether the start and end points are in
the same column. When the two points are in the same column, the process selects (at 8950) text
in the column from the start point to the end point. The process then ends. Otherwise, when the
two points are not in the same column, the process selects (at 8955) text from the start point to
the bottom of the column including the start point, from the top of the column including the end
point to the end point, and all intervening columns. The process then ends. In some
embodiments, the first selected character is the character either at or to the right of the start
point, and the last selected character is the character either at or the left of the end point. In some
embodiments, if the end point is between two lines, then all of the line above is selected and none of the line below is selected. Once the text is selected, the process ends.

[00528]  

Figure 90 illustrates a page 9000 of a semantically reconstructed document. The page 9000 includes two main columns 9005 and 9010 and a centered text box 9015. As shown, these are each assigned a reading order (column 9005 has reading order 1, column 9010 has reading order 2, and text box 9015 has reading order 3). Figure 90 also illustrates a start point 9020 and end point 9025 of a selection, as well as a line 9030 defined by the start and end points. Figure 91 illustrates a selection 9100 defined by start and end points 9020 and 9025 according to process 8900. Even though the line 9030 includes part of text box 9015, because it is a different layout the text is not selected according to process 8900. Instead, the selection 9100 runs from the start point to the end point, selecting all lines in column 9005 below the start point, and all lines in column 9015 above the start point.

[00529]  

Figure 92 also illustrates page 9000, as well as a start point 9220 and end point 9225 of a different selection and the line 9230 defined by the start and end points. Figure 93 illustrates a selection 9300 defined by start and end points 9220 and 9225 according to some embodiments of process 8900. Because the end point 9225 is not in the same layout as the start point 9220, the end point is moved to a point in the same layout as the start point 9220. The embodiment depicted in Figure 93 draws a line between the start and end points and moves the end point to the location where the line leaves the start point’s layout. Selection 9300 is the result of the end point 9225 being moved to point 9305.

[00530]  

Figure 94 illustrates another detailed process 9400 of some embodiments for defining a selection of text in a semantically reconstructed document. Some embodiments implement either process 8900 or process 9400. Other embodiments allow a user to choose (e.g., via a menu option) whether text selections are made using process 8900 (which restricts selections to only one layout) or process 9400 (which allows selections across multiple layouts).

[00531]  

As shown, process 9400 receives (at 9405) zone and layout information for a document. In some embodiments, zone information includes a zone graph, populated with the content of the document, as described above in Section II. Layout information, in some embodiments, includes columns and layouts as described above in Section IV, as well as the
flow of reading through the layouts and between layouts.

[00532] Next, the process assigns (at 9410) a reading order to each column in the document. Some embodiments start at zero on each page. Some embodiments impose the requirement that within a particular layout, the reading order assigned to each column must be consistent with the order of the columns in the layout, and no column outside a particular layout can have a reading order between those of the starting and ending columns of the particular layout. As for which layout comes first in the reading order, that decision is arbitrary in some embodiments. Other embodiments attempt to discern a reading order from layout to layout based on the design of the page.

[00533] Process 9400 then displays (at 9415) a portion of the document. In some embodiments, the document is displayed in its original viewing dimensions on a standard screen. Some embodiments, though, display the document in an adaptive manner using one of the processes described above in Section VIII (e.g., if displaying the document on a small-screen device).

[00534] The process then receives (at 9420) start and end points for a text selection. Some embodiments receive start and end points through a cursor control device (e.g., a mouse), a keyboard, or a combination thereof. For instance, a selection can be defined by a user clicking on a start point in the text with a mouse, holding the mouse button down, dragging to an end point in the text, and releasing the mouse button. Similar processes can be performed with other cursor control devices. Some embodiments allow other sorts of selections with cursor control devices as well - e.g., double-clicking to select a word, or triple-clicking to select a line of text. A user can also, in some embodiments, use selection keys on a keyboard (e.g., shift and arrow keys) to select text in a semantically reconstructed document.

[00535] Process 9400 then defines (at 9425) a line between the start and end point. If either the start or end point (or both) is not in a column (i.e., is in white space, a graphic, etc.), then the process defines (at 9430) a new start point or end point (or both) at the edge of the last column that the line passes through. Some embodiments define this point at the edge of the column where the line passes through the edge. Other embodiments translate the start or end point horizontally into the column to define the new start or end point.
The process then determines (at 9435) whether the start point is before the end point in reading order. In some embodiments, when the start and end point are in the same column, reading order is determined such that the higher (and if tied, left-most) of the two points is the earlier point in the reading order. When the start point is before the end point, the process proceeds to 9445 which is described below. When the start point is after the end point in the reading order, the process switches (at 9440) the start and end point of the selection, such that the start point is always earlier in the reading order than the end point.

Next, the process selects (at 9445) text from the start point to the end of the column that includes the start point, from the start of the column that includes the end point to the end point, as well as all columns in between the starting and ending column in reading order. The process then ends. In some embodiments, the first selected character is the character either at or to the right of the start point, and the last selected character is the character either at or the left of the end point. In some embodiments, when the end point is between two lines, then all of the line above is selected and none of the line below is selected.

Figure 95 illustrates a selection 9500 defined by start and end points 9020 and 9025 (on page 9000 of Figure 90) according to process 9400. Because the start point 9020 and end point 9025 (shown in Figure 90) are in the same layout, the centered text box 9015 is not selected because it is after the end point's column in reading order. A user, dragging along the line 9015, would pass through box 9015, and at that point all of column 9010 and part of text box 9015 would be selected (see below with reference to Figure 96). However, upon the cursor leaving box 9015, only part of column 9010 would be selected up to the location of the cursor (the end point).

Similarly, Figure 96 illustrates a selection 9600 defined by start and end points 9220 and 9225 (on page 9000 of Figure 92) according to process 9400. Because the end point 9225 is in a layout different from the start point 9220, the entire layout of the start point, including most of column 9005 and all of column 9010, is selected. In addition, text in box 9015 is selected up to the end point 9225.

Figures 91 and 93 vs. Figures 95 and 96 illustrate the different selection options available by using either process 8900 or process 9400. One of ordinary skill in the art will
recognize that other selection processes are possible that would use the layout and flow of a
document to arrive at other, possibly beneficial results.

X. EFFICIENT CLUSTER ANALYSIS

As noted in various sections above, some embodiments of the invention utilize
cluster analysis to perform document reconstruction. For instance, alignment guides are
identified with the use of density clustering, joined graphs are identified with the use of bounds
clustering, and gaps between characters are used to identify words and segment gaps with the use
of difference clustering. However, cluster analysis can be very memory-intensive, such that it
can be difficult for a resource-limited device, such as a cell-phone or media player, to perform
cluster analysis.

Accordingly, some embodiments of the invention provide methods for performing
efficient cluster analysis. In some embodiments, the efficient cluster analysis allows cluster
analysis to be performed on a resource-limited device (e.g., a handheld device). Resource-limited
devices can be limited in terms of available memory, processing power, both, or other computing
resources.

In some embodiments, the cluster analysis uses indirectly sorted arrays that stores
indices of an unsorted array. Some embodiments use indirectly sorted arrays to partition data at
multiple different distance scales concurrently, so as to more quickly find an optimal partition of
the data, as opposed to repeating cluster analysis at each different distance scale and comparing
the results.

Figure 97 conceptually illustrates a process 9700 of some embodiments for
semantically reconstructing a document using cluster analysis. As shown, process 9700 receives
(at 9705) a document on a resource-limited device. In some embodiments, the device is a media
player, a cell phone (e.g., an iPhone®), or other handheld device. The document is a vector
graphics document in some embodiments that includes no structural information.

The process then performs (at 9710) efficient cluster analysis on the document
data on the resource-limited device. For instance, some embodiments perform difference
clustering to identify words and segment gaps, density clustering to identify alignment guides,
and bounds clustering to identify compound graphics.
Finally, the process semantically reconstructs (at 9715) the document on the resource-limited device based on the results of the cluster analysis. The process then ends.

**Figure 98** illustrates a sequence 9800 of some embodiments by which a document 9805 is semantically reconstructed on a resource-limited device 9810. The document 9805 is initially parsed (at 9801) into a set 9815 of characters with coordinates. For instance, character 9820 ("r") has coordinates \((X2, Y2)\). Some embodiments also parse graphic objects (e.g., images, shapes, etc.)

Next, efficient cluster analysis is applied (at 9802) to the document data. In some embodiments, this includes using difference clustering to identify words, density clustering to identify guides, and bounds clustering to identify graphs to join. Other reconstruction processes are also performed (at 9803). For instance, paragraphs and columns are identified in some embodiments. One of ordinary skill will recognize that in some embodiments, the cluster analysis processes and other reconstruction processes are not necessarily segregated as far as the order they are performed. The result of the efficient cluster analysis and other reconstruction processes is a semantically reconstructed document 9825 that can be displayed, navigated, etc.

### A. Cluster Analysis as a Set of Operators

Some embodiments perform cluster analysis (whether it be difference clustering, density clustering, or bounds clustering) based on several operators that are applied to sequences of real numbers \((r_1, r_2, \ldots, r_i)\). Some embodiments include the following operators:

- A differencing operator \(D((r_1, r_2, \ldots, r_N)) = (r_v-n, r_3-r_2, \ldots, r_{s-r_2}, \ldots)\). The differencing operator \(D\), in some embodiments, defines a pairwise grouping of the elements \(r_N\) (i.e., defines values for the pairs \(j, r_j\)). \((r_1, r_2, \ldots)\), etc.

- A sorting operator \(S((n, r_2, \ldots, r_N)) = (s_1, s_2, \ldots, s_N)\), where \((s_1, s_2, \ldots, s_N)\) is a permutation of \((n, r_2, \ldots, r_N)\) such that \(s_1 \leq s_2 \leq \ldots \leq s_N\).

- A partitioning operator \(\gamma((r_1, r_2, \ldots, r_N)) = ((\eta_1, \ldots, r_{K+1}), (r_{K+1} = \ldots, r_{K+2}), \ldots, (r_{K+1} = \ldots, r_{K+2}), \ldots)\). In some embodiments, the variable \(g\) is called a gap minimum, and the operator \(P\) partitions the sequence \((n, r_2, \ldots, r_N)\) into non-overlapping subsequences everywhere that the difference between two subsequent values exceeds the gap minimum.
• A coalescing operator $C$ that operates recursively on a partitioned sequence (such as the output of the operator $P$) to join neighboring pairs of subsequences into a single subsequence any number of times. In some embodiments, the tests to determine when to join neighboring pairs are domain-independent.

• A filtering operator $F$ that operates on a partitioned sequence to remove some of the clusters based on tests that are domain independent. The density constraints discussed above in Section III are an example of the use of $F$.

[00549] Some embodiments of difference clustering are performed in terms of the above operators. Similarly, because bounds clustering uses difference clustering with spread values substituted for first-order differences, some embodiments of bounds clustering are performed in terms of the above operators.

[00550] For instance, some embodiments apply the sorting operator $S$ to input data, followed by the difference operator $D$ to generate first-order differences. $S$ and $D$ are then applied to the result data to generate second-order differences (the differences between the differences). The second-order differences are sorted with $S$, and the second-order differences are then split into two disjoint subsequences (the intra-level differences and the larger inter-level differences).

[00551] In some embodiments, the splitting includes further application of $D$ to the second-order differences to obtain third-order differences, followed by $S$ to order the third differences. The split in second-order differences generally occurs where there is one third-order difference substantially larger than the rest. Some embodiments evaluate domain-specific factors as well.

[00552] Once the split is established, some embodiments apply $P$ using a gap minimum equal to the smallest inter-level second difference to partition the ordered first differences, such that each partition represents a level of clustering. Some embodiments apply $C$ to this partition, while some may not. To partition the data into clusters at a particular level, some embodiments apply $P$ to the (sorted) input data using a gap minimum equal to the smallest difference at the particular level. Some embodiments apply $C$ at this point as well, though often with different criteria for coalescing the cluster partition than for the level partitions. Lastly, some
embodiments apply F to disqualify some of the clusters.

[00553] Some embodiments of density clustering are also performed in terms of the above operators. For example, some embodiments apply S followed by D to the input data to generate first-order differences, and apply S to sort the differences. For each of the differences d, some embodiments partition the ordered input data with the operator P using a gap minimum d, then filter the partitions using density constraints. Each of the post-filtering partitions is measured by an optimization metric and the optimal partition is selected as the final clustering. Some embodiments loop through the first-order differences (as gap minimums) starting with the largest and moving to successively smaller values in the sorted sequence.

[00554] In some embodiments, the loop can be ended early for efficiency if there is enough information. Specifically, some embodiments recognize that each successive partition will be the previous partition with one of the clusters split into two clusters. Some embodiments also recognize that clusters that do not meet a minimum size density constraint will never meet such a constraint in the future, so these clusters can be discarded. Once all clusters in a partition have fallen below the minimum size, then the loop is ended prematurely in some embodiments.

B. Efficient Data Structures for Cluster Analysis

[00555] Some embodiments perform efficient cluster analysis by using efficient data structures that allow for memory and processing savings. For instance, when sorting data (e.g., applying the operator S to input data), rather than generating a new array for the data, some embodiments define an array of indices into the array of unsorted data, with the indices sorted in order of the values they reference. This is referred to as an indirectly sorted array in some embodiments. One of ordinary skill will understand that while the examples use arrays, any other suitable data structure may be used as well.

[00556] Figure 99 conceptually illustrates a process 9900 of some embodiments for partitioning a data set by using indirectly sorted arrays. Process 9900 will be described in conjunction with Figure 100. Figure 100 illustrates the partitioning of a data set with nine data items (0.00, 7.43, 17.14, 25.46, 26.60, 30.35, 34.25, 39, and 46.97). As shown in Figure 99, process 9900 receives (at 9905) a sorted array A with data values to be clustered. In some embodiments, the data is character location data for identifying words in a document or
identifying alignment guides. Referring to Figure 100, the data set is stored in a sorted array \( A_{10010} \), with indices \( A[O] - A[H] \).

Next, process 9900 next defines and stores (at 9910) an array \( D(A) \) of first-order differences of the array \( A \) by comparing pairs of subsequent values of array \( A \). In some embodiments, the array \( D(A) \) is generated by use of the operator \( D \) that is described above in subsection A. Figure 100 illustrates the array \( D \) 10015 that stores the first-order differences between the data. For instance, the value in index \( D[3] \) is the value in index \( A[3] \) subtracted from the value in index \( A[4] \) of array \( A \) 10010.

Next, the process defines and stores (at 9915) an indirectly sorted array \( S(D(A)) \) of the indices of \( D(A) \) by applying a sort function to the array \( D(A) \). In some embodiments, the sort function is the operator \( S \) that is described above in subsection A. Figure 100 illustrates the indirectly sorted array \( S(D) \) 10020 that sorts the values of array \( D \) 10015. The first value in the array 10020 ("3") references index 3 of array \( D \) 10015, which is the smallest of the first-order differences ("1.14"). The second value in the array 10020 references index 4 of array \( D \) 10015, which is the second smallest first-order difference, and so on.

The process then determines (at 9920) the minimum size of the gaps between clusters to be used in partitioning the data. In some embodiments, this is the gap minimum \( g \) for use with the partitioning operator \( P \) described above in subsection A. The minimum gap size is specified by a user in some embodiments, or is a value inherent to the problem being solved in others. Some embodiments use multiple partitions (e.g., in the case of density clustering) such that different gap minimums based on the data are used.

Next, process 9900 partitions (at 9925) the data into clusters using consecutive indices stored in the array \( S(D(A)) \). The process then stores (at 9930) the partition. The process then ends. Some embodiments use the indices stored in the indirectly sorted array to partition the data. In some embodiments, the index stored in \( S(D(A)) \) corresponding to the smallest first-order difference that is larger than the gap minimum (i.e., the effective gap minimum) will correspond to the index in the sorted array of data after which the data should be split. All indices stored in the array \( S(D(A)) \) after the effective gap minimum will also indicate where to split the sorted data, because they represent gaps larger than the gap minimum.
Figure 100 illustrates that the effective gap minimum is 7.97 in this example, which is in index 7 in array D 10015. Thus, the partition 10025 of data has four clusters, because it is split in three places (after indexes 7, 2, and 1). Some embodiments store the partition as a single index of the array $S(D(A))$. The partition 10025 is stored as index 10030, which has a value of 5. This indicates that the index corresponding to the effective gap minimum is stored at index 5 of array 10020, and therefore the indices for partitioning the data are stored at indices 5 and up of array 10020.

The above process 9900 enables multiple processing and memory efficiencies for cluster analysis. First, storing the indices (which are integers) rather than the decimal values of the actual data in the sorted array of differences saves memory space. Second, instead of actually storing the partition as multiple separate arrays, it is stored as a single integer value referencing an index of the indirectly sorted array, which can bring about substantial memory savings when there are numerous partitions being evaluated for large arrays of data. Third, the indices at which to partition the data can be read off quickly from the indirectly sorted array, which substantially saves processing time.

These efficiencies can be leveraged in numerous ways to perform cluster analysis. Figure 101 conceptually illustrates a process 10100 of some embodiments for performing cluster analysis at multiple distance scales concurrently. In some embodiments, process 10100 takes advantage of the efficiencies offered by process 9900. As shown, process 10100 defines (at 10105) an indirectly sorted array of differences of data values to be clustered. This is an array such as array 10020 of Figure 100, and is arrived at in some embodiments by sorting the input data values, taking the first-order differences, and then sorting those.

Process 10100 then partitions (at 10110) the data values at several different distance scales concurrently. In some embodiments, this means that multiple partitions are generated for the data using different gap minimums. For instance, in the case of density clustering, each possible partition is generated in some embodiments. In some embodiments, because the first-order differences are sorted with an indirectly sorted array, the partitioning locations for the data can be quickly read off as the indices stored in the indirectly sorted array.

Next, the process stores (at 10115) each partition as an integer value referring to an index of the indirectly sorted array. Integer value 10116 of Figure 101 is an example of
storing a partition as a single integer value. The process then determines (at 10120) the optimal distance scale (and thus the optimal partition). For example, some embodiments use an optimization measure such as is described for density clustering above in Section 111. Furthermore, some embodiments eliminate some of the clusters in a partition by using constraints before testing the partition against the optimization measure.

[00566] Finally, once the optimal distance scale is determined, the process stores (at 10125) the partition of data derived from the optimal distance scale as the set of clusters for the problem being solved. The process then ends. In some embodiments, the set of clusters is stored as a new array once it is determined that it is the optimal set.

[00567] While the above descriptions indicate the efficiencies gained for repeated use of the partitioning operator, the memory and processing efficiencies from indirectly sorted arrays and storing a partition as a single value are applicable to other aspects of cluster analysis as well. For instance, the coalescing operator can take advantage of the same efficiencies in some embodiments.

[00568] As noted above, the coalescing operator C of some embodiments joins neighboring clusters in a partition, possibly repeatedly. The joining of neighboring clusters can be represented as removing a split in a partition. Because each of these splits corresponds to one of the consecutive indices in an indirectly sorted array, coalescing clusters can be defined as disqualifying particular indices from the sequence. As such, the results of applying the coalescing operator to a partition can be a sequence (e.g., an array) of qualifying indices (i.e., indices at which the new partition is split). Storing such a subsequence is much faster in some embodiments than directly moving around the data in the clusters being coalesced.

[00569] Furthermore, coalescing clusters of differences (which is effectively a combination of levels of differences) does not adversely affect the efficiency with which the data clusters (as opposed to the difference clusters) can be quickly read off for a particular chosen level. Even after coalescing the clusters of differences, the indices in the L-th indirectly sorted cluster of differences and above are the split points for the data clusters at level L. The change due to coalescing is that there will be fewer indirectly sorted second differences that determine where each indirectly sorted first difference cluster starts.
[00570] Because the filtering operator (which eliminates clusters of data based on constraints) is only applied to clusters of data (not to clusters of differences), the data clusters have already been determined when the filtering operator is applied, and thus it does not interfere with the efficiencies gained through the above implementations of the partitioning and coalescing operators.

[00571] Efficiencies can also be gained in the splitting of second differences into intra-level and inter-level second differences that is performed in difference clustering, as described above in Section IV. In some embodiments, the conditions used to determine a split point may depend on the clustering of first differences and the data that would result. Thus, the evaluation of these conditions benefits directly from the efficiencies in determining partitions of differences (and thus partitions of data).

[00572] For instance, in the case of difference clustering as applied to document reconstruction, the splitting of second differences is used to determine word breaks and segment breaks (e.g., column, tab, etc. gaps) on a text line, which correspond to first order differences and greater than first order differences respectively. In some embodiments, the goal is to split the second differences such that the minimum of the second cluster of first differences is not much smaller than the expected space character width for the applicable font. Furthermore, a secondary goal would be that the data clusters (each of which is a word) have an average size typical for words in the applicable language. Potential split points can be assessed comparatively lower depending on how far the resulting clusters of first differences and the clusters of data would differ from these expectations. Such assessments can be combined in some embodiments with other measures applied directly to the second differences (e.g., the relative size of the split, the percentile of the split position, and the percentage increase at the split) in a formula that determines the optimal split point. The repeated testing of different splits in the second differences can be made significantly more efficient by the processes described above.

[00573] One of ordinary skill in the art will recognize that while cluster analysis and the specific efficiency techniques described above have primarily been described with respect to its use in document reconstruction, they are applicable to any problem in which there is a set, a distance function on pairs of elements of the set, and a need to identify subsets of elements separated by distances that are small in terms relative to the set. For instance, cluster analysis can
be applied to analyzing user interaction with an application, web page, or video, by clustering position data acquired by measuring eye movements, mouse movements, or touch screen interactions. As another example, a raster image (i.e., bitmap) can be compressed by reducing the number of colors used to encode it. Cluster analysis can be used on the original set of colors to select a reduced set of colors, such that each cluster of colors is replaced by a single color (often equal to an average of its members). Still another example is that some image recognition techniques (e.g., biometrics, optical character recognition, currency validation, etc.) and vectorization of raster images depend on clustering of pixels in a metric space defined by spatial and color coordinate axes. As a final example, patterns in experimental data (e.g., scientific or business data) are often found by plotting data points in a space the axes of which are the parameters of interest. Cluster analysis can be applied to this data, noting that all points in a given cluster have approximately the same values of all parameters of interest.

C. Software Architecture

[00574] In some embodiments, the cluster analysis described above is implemented as software running on a particular machine, such as a computer, a media player, a cell phone (e.g., iPhone®), or other handheld or resource-limited devices (or stored in a computer readable medium). Figure 102 conceptually illustrates the software architecture of a cluster analysis application 10200 of some embodiments for performing cluster analysis. In some embodiments, the application is a stand-alone application or is integrated into another application (e.g., a document reconstruction application), while in other embodiments the application might be implemented within an operating system.

[00575] Cluster analysis application 10200 includes density clustering module 10205, difference clustering module 10210, and bounds clustering module 10215. The application also includes sorting module 10220, differencing module 10225, partitioning module 10230, coalescing module 10235, and filtering module 10240, as well as cluster analysis storage 10245.

[00576] Figure 102 also illustrates document content 10250. One of ordinary skill will recognize that cluster analysis application 10200 could be used for other processes that use cluster analysis that are not related to document reconstruction. Density clustering module 10205, difference clustering module 10210, and bounds clustering module 10215 all receive information (e.g., primitive element position data) from document content 10250. Density
clustering module J0205 performs density clustering as described above in Section III, in part by using the modules 10220-10240. Difference clustering module 10210 performs difference clustering as described above in Section IV, in part by using the modules 10220-10240. Bounds clustering module 10215 performs bounds clustering as described above in Section VI, in part by using the modules 10220-10240. The output of modules 10205-10215 is returned to the document content 10250.

[00577] In some embodiments, the five modules 10220-10240 perform operations associated with the five operators described above in subsection A. The sorting module 10220 of some embodiments receives data from one of the modules 10205-10215 and orders the data (e.g., from lowest value to highest value). The differencing module 10225 of some embodiments receives data from one of the modules 10205-10215 and determines the differences between adjacent pieces of data. The partitioning module 10235 of some embodiments receives data from one of the modules 10205-10215 and partitions the data into multiple subsets. The coalescing module 10235 of some embodiments receives data as multiple subsets from one of the modules 10205-10215 and joins adjacent subsets according to various conditions. The filtering module 10240 of some embodiments receives a partitioned sequence of data in some embodiments and filters out partitions based on various constraints.

[00578] The modules 10220-10240 store data in cluster analysis storage 10245, as well as pass the data back to the modules 10205-10215. In some embodiments, the sorting module 10220 stores its results in cluster analysis storage 10245 as a sorted array of indices (i.e., an indirectly sorted array). The partitioning module, in some embodiments, stores partitions in the cluster analysis storage 10245 as a single integer value referencing an index of an indirectly sorted array.

XI. EFFICIENT DATA STRUCTURES FOR PARSING AND ANALYZING A DOCUMENT

[00579] Some embodiments of the invention provide novel methods and data structures that enable more efficient parsing and analysis of a document. Some embodiments provide an application programming interface (API) that minimizes redundant copies of data as the data is manipulated. An API, in some embodiments, is a set of functions, procedures, methods, classes, or protocols that an operating system, library, service, or framework mnvidcs to sunnort rcnucsts
made by computer programs. In some embodiments, the API is statically linked, while in other embodiments an API is dynamically linked.

[00580] Typically, APIs return copies of internal data or give read-only access to internal data which must then be copied before being manipulated in any way. This creates many layers of redundant data, which slows processing and consumes excess memory. Some embodiments solve this problem by decoupling objects from their data so that object APIs can be made optimal for a programmer at the same time that the data structures are made optimal with respect to performance and memory consumption. Some embodiments use such an API for reconstructing a document as described above in Sections H-X. However, one of ordinary skill in the art will recognize that such an API can be used for any sort of analysis of parsed input data.

[00581] Some embodiments provide an API that appears to a user (e.g., a programmer or a software application using the API) as if the user has their own independent, modifiable copy of the class members of the API with no explicit restrictions. In other words, it appears to the user as though any object returned through the API is completely modifiable by the user. However, in some embodiments, the objects will actually only copy themselves when absolutely necessary, and in most cases will manage memory in such a way as to minimize the amount of memory actually used. The memory management of some embodiments is done by using a sorted array of pointers that has a shared memory object which keeps track of the use of the pointers by other objects. In some embodiments, numerous objects can all reference the same pointer array through the shared memory object, enabling substantial memory savings as compared to making copies of the data at every stage of analysis. One of ordinary skill in the art will recognize that while pointers are used to describe certain features below, any sort of referential data structure could be used.

A. Document Reconstruction with Shared Pointers

[00582] Some embodiments use an API such as is described above to reconstruct a document. Figure 103 conceptually illustrates a process 10300 of some embodiments for reconstructing a document efficiently. Process 10300 will be described in conjunction with Figure 104. Figure 104 illustrates a sequence by which a document 10400 is parsed and analyzed according to process 10300.
As shown in Figure 103, process 10300 receives (at 10305) a portion of a document. In some embodiments, the document portion is a page, and the process operates on a page-by-page basis. In other embodiments, the document portion is an entire document, a section of a document, or a zone on a page. The process then parses (at 10310) the document to determine the characters in the document portion, and stores (at 10315) an array of characters for the parsed data.

Figure 104 illustrates that the document 10400 is parsed into a randomly-ordered array 10405 of characters. While these examples use arrays, one of ordinary skill will understand that any other suitable data structured may be used. In some embodiments, parsing the document involves reading a stream of bytes representing the document and turning that stream into a usable representation (such as the character array) of the information in the stream. The characters in the stream are read in a random order in some embodiments, which is why the order of the array 10405 is random. The characters of some embodiments have coordinates and/or page numbers. In some embodiments, each character is stored as an object that includes the associated coordinate or page number values.

Process 10300 defines (at 10320) a sorted array of pointer that orders the characters for the document portion. In some embodiments, the characters for a page are sorted with a primary sort of top to bottom and a secondary sort of left to right. Some embodiments that store multiple pages in a character array sort by page first. Figure 104 illustrates an array of pointers 10410 that is defined for the sorted characters. The first pointer 10411 points to the letter "L" in the array 10405, the second pointer 10412 to the letter "o", and so on. Defining an array of pointers to the initial character array rather than defining and storing a separate new array saves memory in some embodiments.

The process next receives (at 10325) instructions to manipulate string objects. Some embodiments define a string object as a pointer to a location in the sorted array of pointers and a count of how many characters are in the string. For instance, a string object for the entire page would point to the first pointer in the sorted pointer array (the top-leftmost character), and give a count of the number of characters on the page.

In some embodiments, the instructions include splitting strings, joining strings, adding characters, removing characters, and re-ordering characters.
embodiments, are invoked as part of the process of reconstructing a document and using the reconstructed document as described above in Sections H-X. For instance, in some cases when lines are merged, the order of characters must be modified. When zones are defined, some embodiments define strings for each zone, which in many cases involves splitting strings, joining strings, or both.

[00588] After receiving the instructions, the process determines (at 10330) whether the instructions can be performed using only pointers that are already allocated (e.g., the sorted pointer array defined at 10320). In some embodiments, splitting strings involves only the use of pointers that are already allocated. In the case of document reconstruction, some processes only involve the splitting of strings (e.g., line identification, line splitting, etc.). Furthermore, joining strings that are next to each other in the sorted array of pointers will involve only the use of already-allocated pointers in some embodiments.

[00589] Figure 104 illustrates how identifying the two lines in document 10400 results in two string objects 10415 and 10420 that reference the already-allocated pointers in the array 10410. The first line is defined by a string object 10415 that points to the pointer to L 10411 and has a count of 15 (the number of characters on the first line). The second line is defined by a string object 10420 that points to the pointer to s 10413 and has a count of 7 (the number of characters on the second line). In order to define these lines, no new pointers need to be allocated. Over the hundreds or thousands of operations that may be involved in reconstructing a document, this can introduce large memory and processing time (because no arrays need to be searched) efficiencies.

[00590] The same pointers 10410 can then be used when words are identified. For example, string objects 10425 and 10430 define two of the words in document 10400. These words point to the same start pointers as string objects 10415 and 10420, but have different counts because the words are shorter than the lines. However, no new pointers need to be allocated to define these words, only new string objects. For a full document, hundreds or thousands of different string objects may all reference the same pointer array (such as pointers 10410), introducing large memory savings over repeatedly allocating memory for new pointer arrays.

[00591] When the received instructions can be performed using only those that are
already allocated, process 10300 performs (at 10335) the instructions using the shared pointers that are already allocated in memory. The process then proceeds to 10355, which is described below. Otherwise, the process determines (at 10340) whether the instructions can be performed using a new collection of pointers.

Some embodiments allocate new pointers when instructions cannot be performed with only pointers that are already allocated, but the instructions do not require direct data manipulation of the character array. In some embodiments, joining strings that are not next to each other in a shared array of pointers requires a new allocation of pointers, because a string object for the joined strings cannot be represented by pointing to one pointer in the sorted array and moving forward in that array. For instance, referring to Figure 104, if an operation called for appending the first line to the end of the second line, then string object for the appended lines could not point to array 10410. Instead, a new array of pointers would have to be allocated in the required order.

When the received instructions can be performed using a new allocation of pointers, the process performs (at 10345) the instructions by using a new allocation of pointers to the character array. The process then and proceeds to 10355, which is described below. Otherwise, the process performs (at 10350) the instructions by using a new copy of a portion or the entire character array. Directly editing the document data (i.e., a user adding a word to the document) is an example of instructions that could not be performed without manipulating the actual array of characters in some embodiments. However, a user adding a word to the document would not require a completely new copy, but instead could be handled by adding characters to the array and then defining a new array of pointers to the characters. Similarly, merging text lines often requires a new array of pointers, because a character from one text line may be inserted into the next text line, thereby altering the order of the characters relative to each other.

Next, the process determines (at 10355) whether more instructions to manipulate the string objects have been received. When more instructions have been received, the process proceeds to 10330, which is described above, to determine the most efficient way of performing the instructions. Otherwise, the process ends. This process illustrates the hierarchy of memory and processing savings that are introduced by using the shared data. The original data is shared among various pointer arrays, and each pointer array is shared among many string objects. For
each set of instructions received (e.g., each call into an API), the most efficient way of performing the instructions is used. Ideally, the instructions will not require the creation of any new pointers, and only new string objects need be created. If this is not possible, then memory savings may still be gained by creating new pointers that share the original data as opposed to creating a new character array.

[00595] Although process 10300 has been described with reference to string objects and specifically string objects for document reconstruction, one of ordinary skill in the art will recognize that the efficiencies gained by exhibiting a preference for using already-allocated pointers and then for allocating new pointers as opposed to copying data, are applicable to a wide range of problems where memory and processing time are at a premium.

B. Shared Memory Objects

[00596] In some embodiments, each array of pointers has a shared memory object that manages the use of the pointers in the array. In some embodiments, the shared memory object for a particular pointer array keeps track of the data objects (e.g., string objects) that reference the particular array. In some embodiments, the shared memory object also keeps track of where in memory the pointer array starts as well.

[00597] Figure 105 illustrates the manner in which data is stored according to some embodiments of the invention. Figure 105 illustrates an array of data 10505, a sorted array of pointers 10510, a shared memory object 10515, and data objects 10520. The data array 10505 is randomly-ordered parsed data in some embodiments (e.g., character data from a parsed document).

[00598] The sorted array of pointers 10510 is an array of pointers to the data array 10505. Each pointer points to a data item in the array 10505 in some embodiments. The pointers are arranged in an order based upon a sort of the data. For instance, in the case of a document, the pointers are arranged in the reading order of the characters to which they point in some embodiments.

[00599] Each of the data objects 10520 includes a reference to a location in the pointer array 10510 and a count. The location in the pointer array 10510 for a particular data object is the pointer that points to the first piece of data that the data object references. For instance, when
the data object is a string object for the word "Array", the data object would specify the location 
in the pointer array where the pointer that points to the "A" is found. The data object would also 
include a count of 5.

[00600]  Figure 105 also illustrates a shared memory object 10515. In some embodiments, 
the shared memory object manages the use of the sorted array 105 10 by the data objects 10520. 
The shared memory object 10515 keeps a count of the number of data objects 10520 that 
reference the array 105 10.

[00601]  Some embodiments do not define the shared memory object 10515 when a first 
data object (that points to the start of the array and has a count of the entire array) is defined. 
However, once a second data object points to the array, the array is now shared, and the shared 
memory object 10515 is defined to keep track of how many data objects share the array and 
where the start of the array is, as each individual object does not have this information. 
Accordingly, in some embodiments, the data objects 10520 can call a function to instantiate a 
shared memory object for a pointer array if none exists when the data object is set to point to the 
pointer array. When the number of objects 10520 drops to zero, the shared memory object 10515 
deallocates the pointers 10510 and is then itself removed from memory.

[00602]  In some embodiments, each individual data object 10520 sharing the pointer array 
10510 does not have any knowledge that other objects 10520 are also using the pointers in array 
10510. Furthermore, the objects 10520 do not have any knowledge of the start or end of array 
10510, merely referencing some point in the array 10510. However, the shared memory object 
10520 of some embodiments knows where the start of the array is in memory.

C.  Software Architecture

[00603]  In some embodiments, the API described above are implemented as software 
running on a particular machine, such as a computer, a media player, a cell phone (e.g., an 
iPhone®), or other handheld or resource-limited devices (or stored in a computer readable 
medium). Figure 106 conceptually illustrates an API 10600 that performs document 
reconstruction processes while using the efficiency techniques described above in subsections A 
and B.

[00604]  API 10600 includes geometric analysis modules 10610, document reconstruction
modules 10615, and display and interaction modules 10620. The API 10600 is, in some embodiments, the set of functions, procedures, methods, classes, and/or protocols that is provided for use by external applications 10605.

[00605] The API 10600 receives requests (e.g., function calls) to the public methods by external applications 10605. In some embodiments, there are numerous external applications. For instance, in the case where an API is provided on a handheld device (e.g., an iPhone®), the external applications might be a PDF viewer (e.g., an e-book reader), a word processor (e.g., Microsoft Word, Apple Pages, etc.), a web browser (e.g., Microsoft Internet Explorer, Apple Safari, Mozilla Firefox, etc.), etc.

[00606] The various public methods provided by API 10600 call various private methods that perform the geometric analysis and document reconstruction, access the document object model, etc. The data (e.g., the primitive elements that are initially identified by a parser) is stored in the document reconstruction data 10625. Although it may appear to the external applications that they can access the data (e.g., while manipulating characters to identify words, text lines, etc.), in fact the class members that are manipulated by the external applications through the API are divorced from the actual data by defining the class members to only store references to the data, as described above in subsections A and B.

XII. OVERALL SOFTWARE ARCHITECTURE

[00607] In some embodiments, the processes described above are implemented as software running on a particular machine, such as a computer, a media player, a cell phone (e.g., an LPhonec®), or other handheld or resource-limited devices (or stored in a computer readable medium). Figure 107 conceptually illustrates the software architecture of an application 10700 of some embodiments for reconstructing, displaying, and interacting with a document. In some embodiments, the application is a stand-alone application or is integrated into another application, while in other embodiments the application might be implemented within an operating system. In still other embodiments the modules illustrated in Figure 107 are split among multiple applications. For instance, in some embodiments, one application generates the document object model, while another application displays the document and interacts with the document object model (see full description below).
Application 10700 includes a parser 10710, profiling modules 10720, semantic reconstruction modules 10730, cluster analysis modules 10740, user interaction modules 10750, and display adaptation modules 10760. The application 10700 also includes document data storage 10715, profile storage 10725, cluster analysis storage 10735, and document object module storage 10745. Figure 107 also illustrates an operating system 10770 that includes cursor controller driver 10775, keyboard drive 10780, and display module 10785. In some embodiments, as illustrated, the cursor controller driver 10775, keyboard driver 10780, and/or display module 10785 are part of operating system 10770 even when the compositing application is a stand-alone application separate from the operating system.

As shown, the parser 10710 receives a document 10705. In some embodiments, the document is an unformatted document that includes vector graphics (e.g., a PDF). The parser 10710 parses the document information and stores the parsed data in the document data storage 10715. In some embodiments, the parsed text data is stored as an array of characters as described in Section X above.

The semantic reconstruction modules 10730 reconstruct the document to generate the document object model 10745 from the document data 10715. Semantic reconstruction modules 10730 perform such processes as zone analysis, guide and gutter identification, layout and flow identification, table identification, and joined graph identification.

The output of the semantic reconstruction modules also is sent to the profiling modules 10720. Profiling modules 10720 include a profile matching engine that matches hierarchical profiles and inform the semantic reconstruction modules how to go about performing reconstruction, as described above in Section VII.

The semantic reconstruction modules 10710 also pass information to the cluster analysis modules 10740. Cluster analysis modules 10740 perform density clustering for guide identification, difference clustering for word and segment gap information, and bounds clustering for identifying graphs that should be joined, in some embodiments. The cluster analysis modules use the cluster analysis storage 10735 to store arrays and indices as described in Section X. The results of the cluster analysis are then passed back to the semantic reconstruction modules 10730.
Once the semantic reconstruction modules 10730 have reconstructed the
document, they store the document object model 10745. Document object model 10745 stores all
information about the semantically reconstructed document, such as the zone graph populated
with content that is described above in Section II.

Display adaptation modules 10760 use the document object model 10745 to
determine how to display the document. For instance, display adaptation modules of some
embodiments perform the processes described above in Section VIII for displaying the document
on a small-screen device. Display adaptation modules 10760 pass the display information to the
display module 10785, which governs the actual display on the screen.

User interaction modules 10750 receive input information from the cursor
controller driver 10775 and keyboard driver 10780. The input information directs the user
interaction modules 10750 to perform operations on the document, such as selections as
described above in Section IX, as well as editing of the document. If the document is edited, then
the document object model 10745 must be modified to reflect the edits.

In some embodiments, the results of the processes performed by some of the
above-described modules or other modules are stored in an electronic storage (e.g., as part of a
document object model). The document object model can then be used for displaying the
document on an electronic display device (e.g., a handheld device, computer screen, etc.) such
that a user can review and/or interact with the document (e.g., via touchscreen, cursor control
device, etc.).

Figure 108 conceptually illustrates a process 10800 of some embodiments for
manufacturing a computer readable medium that stores a computer program such as the
application 10700 described above. In some embodiments, the computer readable medium is a
distributable non-volatile electronic storage medium (e.g., CD-ROM, hard disk, device firmware,
etc.).

As shown, process 10800 begins by defining (at 10805) geometric analysis
modules, such as modules 110 of Figure 1. More detailed examples of such modules include the
border identification module 1705, interval and intersection identification module of 1710, zone
identification module 1715, and zone graph builder 1720 of Figure 17. These modules, in some
embodiments, identify boundaries between sets of primitive elements and identify regions bounded by the boundaries. In some embodiments, the modules identify graphical primitive elements as potential boundaries, identify a portion of the potential boundaries as actual boundaries, traverse the actual boundaries to identify zones, and define a hierarchical document module with the identified zones.

[00619] The process then defines (at 10810) document reconstruction modules such as modules 120 of Figure 1. In some embodiments, semantic reconstruction modules 10730 of Figure 107 include both geometric analysis modules and document reconstruction modules, though other embodiments only include one or the other.

[00620] Process 10800 then defines (at 10815) a set of hierarchical profiles, such as profiles 10725. Next, the process defines (at 10820) a set of modules for performing clueter analysis. The cluster analysis modules 10740 are an example of such modules. The process then defines (at 10825) modules for adaptively displaying a document, such as display adaptation modules 10760. Next, process 10800 defines (at 10830) modules for receiving user interactions with a document, such as modules 10750.

[00621] The process also defines (at 10835) other modules. For instance, some embodiments include modules for parsing an incoming document (e.g., a document received by the application) or for efficiently using memory and processing time when performing various document reconstruction operations.

[00622] Process 10800 then stores (at 10840) the application on a computer readable storage medium. As mentioned above, in some embodiments the computer readable storage medium is a distributable CD-ROM. In some embodiments, the medium is one or more of a solid-state device, a hard disk, a CD-ROM, or other non-volatile computer readable storage medium. The medium may be firmware of a handheld device (e.g., an iPhonc®) in some embodiments.

[00623] One of ordinary skill in the art will recognize that the various elements defined by process 10800 are not exhaustive of the modules, rules, and processes that could be defined and stored on a computer readable storage medium for an application incorporating some embodiments of the invention. Furthermore, it is equally possible that some embodiments will
include only a subset of the elements defined by process 10800 rather than all of them.

[00624] In addition, the process 10800 is a conceptual process, and the actual implementations may vary. For example, different embodiments may define the various elements in a different order, may define several elements in one operation, may decompose the definition of a single element into multiple operations, etc. Furthermore, the process 10800 may be implemented as several sub-processes or combined with other operations in a macro-process.

XIII. COMPUTER SYSTEM

[00625] Many of the above-described features and applications are implemented as software processes that are specified as a set of instructions recorded on a computer readable storage medium (also referred to as computer readable medium). When these instructions are executed by one or more computational element(s) (such as processors or other computational elements like ASICs and FPGAs), they cause the computational element(s) to perform the actions indicated in the instructions. Computer is meant in its broadest sense, and can include any electronic device with a processor. Examples of computer readable media include, but are not limited to, CD-ROMs, flash drives, RAM chips, hard drives, EPROMs, etc. The computer readable media does not include carrier waves and electronic signals passing wirelessly or over wired connections.

[00626] In this specification, the term "software" is meant to include firmware residing in read-only memory or applications stored in magnetic storage which can be read into memory for processing by a processor. Also, in some embodiments, multiple software inventions can be implemented as sub-parts of a larger program while remaining distinct software inventions. In some embodiments, multiple software inventions can also be implemented as separate programs. Finally, any combination of separate programs that together implement a software invention described here is within the scope of the invention. In some embodiments, the software programs when installed to operate on one or more computer systems define one or more specific machine implementations that execute and perform the operations of the software programs.

[00627] Figure 109 illustrates a computer system with which some embodiments of the invention are implemented. Such a computer system includes various types of computer readable media and interfaces for various other types of computer readable media. Computer system
10900 includes a bus 10905, a processor 10910, a graphics processing unit (GPU) 10920, a system memory 10925, a read-only memory 10930, a permanent storage device 10935, input devices 10940, and output devices 10945.

[00628] The bus 10905 collectively represents all system, peripheral, and chipset buses that communicatively connect the numerous internal devices of the computer system 10900. For instance, the bus 10905 communicatively connects the processor 10910 with the read-only memory 10930, the GPU 10920, the system memory 10925, and the permanent storage device 10935.

[00629] From these various memory units, the processor 10910 retrieves instructions to execute and data to process in order to execute the processes of the invention. In some embodiments, the processor comprises a Field Programmable Gate Array (FPGA), an ASIC, or various other electronic components for executing instructions. Some instructions are passed to and executed by the GPU 10920. The GPU 10920 can offload various computations or complement the image processing provided by the processor 10910. In some embodiments, such functionality can be provided using Corclmage's kernel shading language.

[00630] The read-only-memory (ROM) 10930 stores static data and instructions that are needed by the processor 10910 and other modules of the computer system. The permanent storage device 10935, on the other hand, is a read-and-write memory device. This device is a non-volatile memory unit that stores instructions and data even when the computer system 10900 is off. Some embodiments of the invention use a mass-storage device (such as a magnetic or optical disk and its corresponding disk drive) as the permanent storage device 10935.

[00631] Other embodiments use a removable storage device (such as a floppy disk, flash drive, or ZIP® disk, and its corresponding disk drive) as the permanent storage device. Like the permanent storage device 10935, the system memory 10925 is a read-and-write memory device. However, unlike storage device 10935, the system memory is a volatile read-and-write memory, such a random access memory. The system memory stores some of the instructions and data that the processor needs at runtime. In some embodiments, the invention's processes are stored in the system memory 10925, the permanent storage device 10935, and/or the read-only memory 10930. For example, the various memory units include instructions for processing multimedia items in accordance with some embodiments. From these various memory units, the processor
10910 retrieves instructions to execute and data to process in order to execute the processes of some embodiments.

[00632] The bus 10905 also connects to the input and output devices 10940 and 10945. The input devices enable the user to communicate information and select commands to the computer system. The input devices 10940 include alphanumeric keyboards and pointing devices (also called "cursor control devices"). The output devices 10945 display images generated by the computer system. The output devices include printers and display devices, such as cathode ray tubes (CRT) or liquid crystal displays (LCD).

[00633] Finally, as shown in Figure 109, bus 10905 also couples computer 10900 to a network 10965 through a network adapter (not shown). In this manner, the computer can be a part of a network of computers (such as a local area network ("LAN"), a wide area network ("WAN"), or an Intranet, or a network of networks, such as the internet. Any or all components of computer system 10900 may be used in conjunction with the invention.

[00634] Some embodiments include electronic components, such as microprocessors, storage and memory that store computer program instructions in a machine-readable or computer-readable medium (alternatively referred to as computer-readable storage media, machine-readable media, or machine-readable storage media). Some examples of such computer-readable media include RAM, ROM, read-only compact discs (CD-ROM), recordable compact discs (CD-R), rewritable compact discs (CD-RW), read-only digital versatile discs (e.g., DVD-ROM, dual-layer DVD-ROM), a variety of recordable/rewritable DVDs (e.g., DVD-RAM, DVD-RW, DVD+RW, etc.), flash memory (e.g., SD cards, mini-SD cards, micro-SD cards, etc.), magnetic and/or solid state hard drives, read-only and recordable blu-ray discs, ultra density optical discs, any other optical or magnetic media, and floppy disks. The computer-readable media may store a computer program that is executable by at least one processor and includes sets of instructions for performing various operations. Examples of hardware devices configured to store and execute sets of instructions include, but are not limited to application specific integrated circuits (ASICs), field programmable gate arrays (FPGA), programmable logic devices (PLDs), ROM, and RAM devices. Examples of computer programs or computer code include machine code, such as is produced by a compiler, and files including higher-level code that are executed by a computer, an electronic component, or a microprocessor using an
interpreter.

[00635] As used in this specification and any claims of this application, the terms "computer", "server", "processor", and "memory" all refer to electronic or other technological devices. These terms exclude people or groups of people. For the purposes of the specification, the terms display or displaying means displaying on an electronic device. As used in this specification and any claims of this application, the terms "computer readable medium" and "computer readable media" are entirely restricted to tangible, physical objects that store information in a form that is readable by a computer. These terms exclude any wireless signals, wired download signals, and any other ephemeral signals.

[00636] While the invention has been described with reference to numerous specific details, one of ordinary skill in the art will recognize that the invention can be embodied in other specific forms without departing from the spirit of the invention. For example, some embodiments receive a document in which each page is defined as a single image. However, some embodiments can perform optical character recognition on the document to recognize glyphs, and in some cases shapes (e.g., lines, rectangles, etc.), after which point the document can be reconstructed. Also, some embodiments have been described above as performing particular geometric analysis and document reconstruction operations on particular primitive elements. However, one of ordinary skill would recognize that the operations could be applied to other sorts of primitive elements. For instance, guide identification is described as involving the use of density clustering to identify associations of (i.e., to associate, or to define associations of) glyphs forming a vertical boundary. However, similar operations could be applied to look for clusters of primitive shapes that form boundaries (e.g., dashed lines).

[00637] Furthermore, a number of the figures (including Figures 3, 8, 9, 12, 15, 18, 20, 21, 25, 26, 31, 33, 36, 38, 40, 42, 48, 53, 55, 57, 62, 64, 66, 68, 71, 72, 78, 80-82, 84, 85, 87, 89, 94, 97, 99, 101, 103, and 108) conceptually illustrate processes. The specific operations of these processes may not be performed in the exact order shown and described, the specific operations may not be performed in one continuous series of operations, and different specific operations may be performed in different embodiments. Furthermore, the process could be implemented using several sub-processes, or as part of a larger macro process. Thus, one of ordinary skill in the art would understand that the invention is not to be limited by the foregoing illustrative
details, but rather is to be defined by the appended claims.
CLAIMS

1. A computer readable medium storing a computer program which when executed by at least one processor analyzes a document that comprises a plurality of primitive elements, the computer program comprising sets of instructions for:
   identifying boundaries between sets of primitive elements;
   identifying regions bounded by the boundaries; and
   defining a structured document based on the regions and the primitive elements.

2. The computer readable medium of claim 1, wherein the set of instructions for defining a structured document comprises a set of instructions for identifying a hierarchical relationship between at least two identified regions.

3. The computer readable medium of claim 2, wherein a first region that is wholly within a second region is a child of the second region.

4. The computer readable medium of claim 2, wherein the set of instructions for identifying a hierarchical relationship between two identified regions includes a set of instructions for defining a hierarchical document object model in which each identified region is a node of the document object model.

5. The computer readable medium of claim 4, wherein the set of instructions for defining the structured document further comprises a set of instructions for populating the document object model with at least one structural element defined from the primitive elements.

6. The computer readable medium of claim 1, wherein boundaries are identified based on locations of particular primitive elements in the document.

7. The computer readable medium of claim 6, wherein the particular primitive elements are straight lines or shapes that approximate straight lines.

8. The computer readable medium of claim 1, wherein the set of instructions for identifying the regions comprises sets of instructions for:
   selecting a set of boundaries that actually border a regions based on whether the boundaries intersect other boundaries; and
   traversing a subset of the boundaries to identify a region.

9. The computer readable medium of claim 1, wherein the set of instructions for identifying boundaries between sets of primitive elements comprises sets of instructions for:
   identifying, as potential boundaries, primitive elements and groups of primitive
elements that are straight lines or shapes that approximate straight lines;
identifying intersections between said potential boundaries; and
removing potential boundaries that do not intersect at least two other potential boundaries.

10. The computer readable medium of claim 1, wherein the set of instructions for identifying regions bounded by the boundaries comprises sets of instructions for traversing the identified boundaries to identify closed loops of boundaries, wherein each closed loop is a region.

11. The computer readable medium of claim 1, wherein the computer program further comprises a set of instructions for using the identified regions to define structural elements for the document, wherein the structured document is further based on the structural elements.

12. The computer readable medium of claim 11, wherein the set of instructions for using the identified regions to define structural elements comprises a set of instructions for analyzing each particular region separately to create associations between sets of primitive elements in the particular region.

13. A method for defining a program for (i) analyzing a document that comprises a plurality of primitive elements and (ii) generating structural elements that define structure in said document based on said analysis, the method comprising:
   - defining a module for identifying boundaries between sets of primitive elements;
   - defining a module for identifying regions bounded by the boundaries;
   - defining a module for using the identified boundaries and regions to specify the structural elements.

14. The method of claim 13 further comprising defining a module for defining a structured document based on the regions, the primitive elements, and the structural elements.

15. The method of claim 13, wherein the document is an unstructured document.

16. A computer readable medium storing a computer program which when executed by at least one processor analyzes a document that comprises a plurality of primitive elements, the primitive elements comprising a plurality of glyphs and a plurality of graphical elements, the computer program comprising sets of instructions for:
   - identifying a plurality of the graphical elements as potential boundaries;
   - identifying a portion of the potential boundaries as actual boundaries;
traversing the actual boundaries to identify one or more zones; and
defining a hierarchical document model with the identified zones.

17. The computer readable medium of claim 16, wherein identifying a plurality of the
graphical elements as potential boundaries comprises sets of instructions for:
   identifying graphical elements that are vertical and horizontal straight lines of at
   most a particular thickness;
   identifying graphical elements that approximate vertical and horizontal straight
   lines of at most a particular thickness; and
   identifying outer edges of upright rectangular graphical elements.

18. The computer readable medium of claim 16, wherein the set of instructions for
   identifying a portion of the potential boundaries as actual boundaries comprises sets of
   instructions for:
      identifying intersections between the potential boundaries; and
      iteratively removing potential boundaries that do not intersect at least two other
      potential boundaries until all remaining potential boundaries intersect at least two other potential
      boundaries; and
   identifying the remaining potential boundaries as actual boundaries.

19. The computer readable medium of claim 16, wherein the set of instructions for
   traversing the actual boundaries to identify one or more zones comprises sets of instructions for:
      defining a set of directionally-aligned intervals comprising two intervals aligned
      in opposite directions for each of the actual boundaries;
      selecting an intersection and a direction away from the intersection;
      traversing the intervals in the set of intervals beginning from the selected
      intersection in the selected direction until returning to the selected intersection;
      defining the area enclosed by the traversed intervals as a zone; and
      removing the traversed intervals from the set of intervals.

20. The computer readable medium of claim 19, wherein the directionally-aligned
    intervals are vectors.

21. The computer readable medium of claim 19, wherein the set of instructions for
    traversing the actual boundaries to identify one or more zones further comprises sets of
    instructions for:
repeatedly, until the set of directionally-aligned intervals is empty:
selecting an intersection and a direction from the intersection;
traversing the intervals in the set of intervals beginning from the selected
intersection in the selected direction until returning to the selected intersection;
defining the area enclosed by the traversed intervals as a zone; and
removing the traversed intervals from the set of intervals.

22. The computer readable medium of claim 16, further comprising defining a
particular zone as an island when the particular zone is traversed in a particular direction.

23. The computer readable medium of claim 16, wherein a first zone is a parent of a
second zone when the second zone is fully enclosed by the first zone.

24. The computer readable medium of claim 23, wherein the second zone is an island
when the first zone and second zone do not share a boundary.

25. The computer readable medium of claim 16, wherein the computer program
further comprises a set of instructions for defining structural elements based on the glyphs and
the graphical elements that are not actual boundaries.

26. The computer readable medium of claim 25, wherein the computer program
further comprises a set of instructions for populating a zone of the hierarchical document model
with at least one structural element.

27. The computer readable medium of claim 25, wherein the computer program
further comprises a set of instructions for identifying a rotation zone.

28. The computer readable medium of claim 27, wherein a rotation zone is a zone that
is not aligned with a page that includes the rotation zone.

29. The computer readable medium of claim 28, wherein a zone is not aligned with a
page when the zone is offset from edges of the page by at least a particular angle.

30. A computer readable medium storing a computer program which when executed
by at least one processor analyzes a document that comprises a plurality of words, each word
comprising an associated set of glyphs, each glyph having location coordinates, the computer
program comprising sets of instructions for:
identifying a group of words based on the location coordinates of the glyphs;
based on the identified group of words, defining a set of boundary elements for
the glyphs;
defining a structured document based on the glyphs and the defined set of boundary elements.

31. The computer readable medium of claim 30, wherein the computer program further comprises a set of instructions for defining at least one region of empty space between a pair of boundary elements, wherein the structured document is further defined based on the region of empty space.

32. The computer readable medium of claim 30, wherein the set of instructions for identifying the groups of words comprises a set of instructions for performing cluster analysis on data derived from coordinates of the words along one axis.

33. The computer readable medium of claim 32, wherein the cluster analysis comprises density clustering in order to identify large groups of data with similar values.

34. The computer readable medium of claim 30, wherein the boundary elements are right and left alignment guides for glyphs.

35. The computer readable medium of claim 30, wherein the set of instructions for defining the set of boundary elements comprises sets of instructions for:

   for each cluster, defining a strip along a first axis with a width based on the words in the cluster; and

   eliminating the strip at coordinates along the first axis where a word crosses into the strip from a particular direction along a second axis.

36. The computer readable medium of claim 30, wherein the set of instructions for defining a structured document comprises a set of instructions for identifying a column of text using the glyphs and the defined boundary elements.

37. A method for defining a program for (i) analyzing a document that comprises a plurality of words, each word comprising an associated set of glyphs, each glyph having location coordinates, and (ii) generating structural elements that define structure in said document based on said analysis the method comprising:

   defining a module for identifying groups of words based on the location coordinates of the glyphs;

   defining a module for defining a set of boundary elements for the glyphs based on the identified groups of words, the boundary elements identifying a set of borders for the glyphs;

   and
defining a module for using the identified boundary elements to specify the structural elements

38. The method of claim 37 further comprising defining a module for defining a structured document based on the glyphs the defined boundary elements, and the structural elements.

39. A computer readable medium storing a computer program which when executed by at least one processor analyzes a document comprising a plurality of words, each word comprising a plurality of glyphs, each word having a particular location value, the computer program comprising sets of instructions for:

- ordering the location values of the words;
- identifying a plurality of different groupings of the location values into subsets.
- for each different grouping, identifying a set of the subsets of location values that satisfy a particular set of constraints;
- determining a particular one of the sets of the subsets of location values that optimizes a particular measure; and
- defining boundary elements for the glyphs based on the particular set of subsets that optimizes the particular measure.

40. The computer readable medium of claim 39, wherein the set of constraints includes a requirement that the subsets of location values have at least a particular number of values.

41. The computer readable medium of claim 39, wherein the set of constraints includes a requirement that the difference between the largest and smallest value in the subset be no more than a particular threshold.

42. The computer readable medium of claim 39, wherein the particular location values represent right edges of words, wherein the set of subsets that optimizes the particular measure is used to identify right alignment guides.

43. The computer readable medium of claim 39, wherein the particular location values represent left edges of words, wherein the set of subsets that optimizes the particular measure is used to identify left alignment guides.

44. A computer readable medium storing a computer program which when executed by at least one processor identifies clusters of data, the computer program comprising sets of
instructions for:

receiving a set of data values to be clustered;

identifying a plurality of partitions of the data values, each different partition specifying a different grouping of the data values into subsets;

for each group of subsets of data values, identifying a set of the subsets that satisfy a particular set of constraints; and

determining a set of subsets that optimizes a particular measure.

45. The computer readable medium of claim 44, wherein the data values are positions of glyphs in a document, wherein the set of subsets that optimizes a particular measure identifies alignment guides for the document.

46. A computer readable medium storing a computer program which when executed by at least one processor analyzes a document that comprises a plurality of words, each word an associated set of glyphs, the computer program comprising sets of instructions for:

identifying a set of left alignment points and a set of right alignment points in the document;

identifying an empty space in the document between a left alignment point and a right alignment point that satisfies particular criteria; and

defining a structured document using the identified empty space.

47. The computer readable medium of claim 46, wherein the set of left alignment points and the set of right alignment points comprise alignment guides determined by density clustering of words.

48. The computer readable medium of claim 46, wherein the set of instructions for identifying an empty space in the document comprises sets of instructions for:

selecting a right alignment point from the set of right alignment points;

identifying a left alignment point in the set of left alignment points that is the closest left alignment point to the selected right alignment point that is right of the selected right alignment point;

defining a vertical strip between the selected right alignment point and the identified left alignment point; and

eliminating the vertical strip at coordinates where a word crosses into the strip.

49. The computer readable medium of claim 44, wherein the set of instructions for
defining a structured document using the identified empty space comprises sets of instructions for splitting lines of words.

50. A computer readable medium storing a computer program which when executed by at least one processor analyzes a document comprising a plurality of glyphs, each glyph having a position in the document, the computer program comprising sets of instructions for:

   based on positions of the glyphs in the document, creating associations between glyphs to identify different sets of glyphs as different words;

   creating associations between words to identify different sets of words as different paragraphs; and

   defining an association between at least two paragraphs that are not contiguous in order to define a reading order for the paragraphs.

51. The computer readable medium of claim 50, wherein the set of instructions for creating associations between words comprises a set of instructions for identifying a set of words to associate based on (i) vertical spacing properties of the glyphs that form the words and (ii) horizontal alignment properties of the glyphs that form the words.

52. The computer readable medium of claim 50, wherein the set of instructions for creating associations between words in order to identify different sets of words as different paragraphs comprises sets of instructions for:

   creating an association between words to identify a set of words as a text line; and

   creating an association between text lines to identify a set of text lines as a paragraph.

53. The computer readable medium of claim 52, wherein the set of instructions for creating an association between words to identify a set of words as a text line comprises sets of instructions for:

   identifying a set of horizontally aligned words, wherein all the words in the set have baselines within a particular threshold of each other; and

   splitting the set of horizontally aligned words based on the presence of at least one horizontal gap; and

   creating associations between the words in each remaining set.

54. The computer readable medium of claim 53, wherein the set of instructions for splitting the set of horizontally aligned words comprises sets of instructions for:
identifying horizontal gaps as gaps of at least a threshold width between horizontally aligned words;
identifying sets of horizontal gaps that are vertically aligned;
removing sets of horizontal gaps that do not meet a particular set of criteria; and
splitting the set of horizontally aligned words at the unremoved gaps.

55. The computer readable medium of claim 52, wherein the set of instructions for creating an association between text lines to identify a set of text lines as a paragraph comprises sets of instructions for comparing vertical spacing properties of adjacent text lines.

56. The computer readable medium of claim 52, wherein the set of instructions for creating an associations between text lines to identify a set of text lines as a paragraph comprises sets of instructions for comparing horizontal alignment properties of adjacent text lines.

57. The computer readable medium of claim 50, wherein the computer program further comprises a set of instructions for creating an associations between paragraphs to identify a sets of paragraphs as a column.

58. A method for defining a program for analyzing a document comprising a plurality of glyphs, each glyph having a position in the document, the method comprising:
defining a module for, based on positions of the glyphs in the document, creating associations between glyphs to identify different sets of glyphs as different words;
defining a module for creating associations between words to identify different sets of words as different paragraphs; and
defining a module for defining an association between at least two paragraphs that are not contiguous in order to define a reading order for the paragraphs.

59. The method of claim 58, wherein defining the module for creating an association between glyphs comprises defining a module for performing cluster analysis on the positions of the glyphs to identify horizontal spacing between glyphs.

60. The method of claim 59, wherein the cluster analysis identifies clusters of horizontal spacing sizes to identify spacing between words and spacing within words.

61. A computer readable medium storing a computer program which when executed by at least one processor analyzes a document comprising a plurality of glyphs, each glyph having a position in the document, the computer program comprising sets of instructions for:
based on the positions of the glyphs, identifying different sets of glyphs as
different words;
  identifying different sets of words as different paragraphs;
  defining a reading order through the paragraphs, said reading order specifying a flow through at least two paragraphs that are not contiguous in the document; and
  defining a structured document based on the words, paragraphs, and the defined reading order.

62. The computer readable medium of claim 61, wherein identifying a particular set of glyphs as a word comprises associating the glyphs.

63. The computer readable medium of claim 61, wherein identifying a particular set of glyphs as a word comprises creating an association between the glyphs.

64. The computer readable medium of claim 61, wherein the two paragraphs that are not contiguous comprise a first paragraph at the bottom of a first column and a second paragraph at the top of a second column, wherein the reading order specifies a flow from the first paragraph directly to the second paragraph.

65. The computer readable medium of claim 61, wherein the two paragraphs that are not contiguous comprise a first paragraph at the end of a first page and a second paragraph at the start of a second page, wherein the reading order specifies a flow from the first paragraph directly to the second paragraph.

66. The computer readable medium of claim 65, wherein the first page and the second page are not contiguous in the document.

67. A computer readable medium storing a computer program which when executed by at least one processor analyzes a document comprising a plurality of primitive elements, each primitive element having a plurality of properties, the computer program comprising sets of instructions for:

  associating sets of primitive elements as structural elements based on the properties of the primitive elements;
  defining relationships between the structural elements;
  specifying a flow through all of the primitive elements based on the associations and relationships; and
  optimizing user interactions with the document by using the specified flow through the primitive elements.
68. The computer readable medium of claim 67, wherein the user interactions comprise selection operations.

69. The computer readable medium of claim 67, wherein the user interactions comprise display operations.

70. The computer readable medium of claim 67, wherein the user interactions comprise navigation operations.

71. A computer readable medium storing a computer program which when executed by at least one processor analyzes a document comprising a plurality of primitive elements, the computer program comprising sets of instructions for:
   identifying boundaries between sets of primitive elements;
   identifying that a plurality of the boundaries form a table; and
   defining a tabular structural element for the table, the tabular structural element comprising a plurality of cells arranged in a plurality of rows and columns, each cell comprising an associated set of primitive elements.

72. The computer readable medium of claim 71 further comprising a set of instructions for defining a structured document based on the tabular structural element.

73. The computer readable medium of claim 72, wherein the set of instructions for defining the structured document comprises a set of instructions for defining a hierarchical model of the unstructured document.

74. The computer readable medium of claim 73, wherein the cells of the tabular structural element comprise nodes in the hierarchical model.

75. The computer readable medium of claim 73, wherein the tabular structural element comprises a node in the hierarchical model.

76. The computer readable medium of claim 71, wherein the set of instructions for identifying that a plurality of the boundaries form a table comprises sets of instructions for identifying (i) a set of boundaries that form a larger rectangular shape and (ii) a plurality of rectangular shapes contained within the larger rectangular shape.

77. The computer readable medium of claim 71, wherein the document comprises a plurality of graphic primitive elements, wherein the set of instructions for identifying boundaries between sets of primitive elements comprises identifying graphic primitive elements that are vertical or horizontal lines and graphic primitive elements that approximate vertical or horizontal
78. The computer readable medium of claim 77, wherein the computer program further comprises a set of instructions for identifying a plurality of hierarchical regions based on said identified graphic primitive elements.

79. The computer readable medium of claim 78, wherein the set of instructions for identifying that a plurality of the boundaries form a table comprises a set of instructions for identifying a set of the hierarchical regions that satisfy a particular set of characteristics.

80. The computer readable medium of claim 79, wherein the set of instructions for identifying that a plurality of the boundaries form a table comprises a set of instructions for identifying (i) a first rectangular region that does not share any borders with a parent region and (ii) a set of at least two child regions of the first region, wherein the set of child regions entirely partitions the first region, wherein each child region in the set is rectangular.

81. The computer readable medium of claim 80, wherein the first region is an outer bound of the table and the child regions of the first region are cells of the table.

82. The computer readable medium of claim 71, wherein a plurality of the identified boundaries are inferred based on positions of the associated sets of glyphs that form the cells.

83. The computer readable medium of claim 82, wherein the computer program further comprises a set of instructions for associating sets of glyphs as paragraphs, wherein the boundaries are inferred based on positions of the paragraphs.

84. The computer readable medium of claim 82, wherein the set of instructions for identifying boundaries between sets of glyphs comprises a set of instructions for identifying at least two sets of glyphs that are vertically or horizontally aligned.

85. The computer readable medium of claim 71, wherein the set of instructions for defining a tabular structural element comprises a set of instructions for identifying sets of identified boundaries that form contiguous rectangles.

86. The computer readable medium of claim 85, wherein the contiguous rectangles are the cells of the tabular structural element.

87. A method for defining a program for (i) analyzing a document comprising a plurality of primitive elements and (ii) generating structural elements that define structure in said document based on said analysis, the method comprising:

   defining a module for identifying boundaries between sets of primitive elements;
defining a module for identifying that a plurality of the boundaries form a table; and

defining a module for defining a tabular structural element based on the table, the tabular structural element comprising a plurality of cells arranged in a plurality of rows and columns, each cell comprising an associated set of primitive elements.

88. A computer readable medium storing a computer program which when executed by at least one processor analyzes a document comprising a plurality of primitive elements, the computer program comprising sets of instructions for:

identifying a first set of primitive elements that comprise a table;
defining a tabular structural element for the first set of primitive elements;
identifying a second set of primitive elements that do not comprise a table;
defining a set of non-tabular structural elements for the second set of primitive elements;

defining a structured document comprising the tabular structural element and the set of non-tabular structural elements.

89. The computer readable medium of claim 88, wherein the set of instructions, for defining a structured document comprises a set of instructions for defining a reading order through the structural elements, wherein the reading order flows from a first structural element from the set of structural elements to the tabular structural element to a second structural element from the set of structural elements.

90. The computer readable medium of claim 88, wherein the computer program further comprises sets of instructions for:

identifying different sets of primitive elements, wherein each of the different sets comprises a different table; and
defining different tabular structural elements for each of the different sets.

91. The computer readable medium of claim 90, wherein a first one of the different sets comprises both graphical primitive elements and glyphs while a second one of the different sets comprises only glyphs.

92. A computer readable medium storing a computer program which when executed by at least one processor analyzes a document, the computer program comprising sets of instructions for:
receiving the document that comprises a plurality of primitive graphic elements, each primitive graphic element defined as a single object in the document, the document having a drawing order that indicates the order in which the primitive graphic elements are drawn when the document is displayed;

identifying positional relationships between successive primitive graphic elements in the drawing order; and

based on the positional relationships, defining a single structural graphic element from at least two of the primitive graphic elements.

93. The computer readable medium of claim 92, wherein the document further comprises a plurality of glyphs, wherein the drawing order does not indicate the order in which the glyphs are drawn on the page.

94. The computer readable medium of claim 92, wherein the document is a vector graphics document, wherein the primitive graphic elements are vector graphics.

95. The computer readable medium of claim 92, wherein the set of instructions for identifying positional relationships between successive primitive graphic elements comprises a set of instructions for calculating, for a first primitive graphic element and a second primitive graphic element that is subsequent to the first in the reading order, a size of a structural graphic element that comprises the first and second primitive graphic elements.

96. The computer readable medium of claim 95, wherein the size is proportional to the area of an upright bounding box that envelops the first and second primitive graphic elements.

97. The computer readable medium of claim 95, wherein the size is proportional to the sum of the length and height of an upright bounding box that envelops the first and second primitive graphic elements.

98. The computer readable medium of claim 95, wherein the set of instructions for identifying positional relationships between successive primitive graphic elements further comprises sets of instructions for:

calculating, for the second primitive graphic element and a third primitive graphic element that is subsequent to the second in the reading order, a size of a structural graphic element that comprises the second and third primitive graphic elements.

comparing the size of the structural graphic element comprising the first and
second primitive graphic elements to the size of the structural graphic element comprising the
second and third graphic elements.

99. The computer readable medium of claim 92, wherein the set of instructions for
identifying positional relationships between successive primitive graphic elements comprises
sets of instructions for:

identifying each pair of primitive graphic elements that are successive in the
drawing order; and

for each identified pair of primitive graphic elements, calculating a spread value
for the pair.

100. The computer readable medium of claim 99, wherein the spread value of a pair
describes the size and proximity of the two primitive graphic elements in the pair.

101. The computer readable medium of claim 99, wherein the set of instructions for
defining a single structural graphic element from a set of primitive graphic elements comprises
sets of instructions for:

identifying a clusters of primitive graphic elements that are near each other based
on the calculated spread values, wherein within the cluster all of the primitive graphic elements
are consecutive in the drawing order;

identifying sub-groups of consecutive primitive graphic elements in the cluster
that satisfy a particular set of constraints; and

for each sub-group, defining a structural graphic element comprising all of the
primitive graphic elements in the sub-group.

102. The computer readable medium of claim 101, wherein the set of instructions for
identifying the cluster of primitive graphic elements comprises a set of instructions for
performing difference clustering on the primitive graphic elements with the calculated spread
values as first-order differences.

103. The computer readable medium of claim 101, wherein the particular set of
constraints comprises a requirement that bounds for each particular primitive element in the sub-
group intersect collective bounds for all primitive elements in the sub-group previous to the
particular primitive element in the drawing order.

104. The computer readable medium of claim 103, wherein the bounds for each
primitive element is a smallest upright bounding box that includes the primitive element.
105. The computer readable medium of claim 103, wherein the bounds for each primitive element is a smallest-area path that completely encloses the primitive element.

106. The computer readable medium of claim 92, wherein the set of instructions for defining the single structural graphic element from the plurality of primitive graphic elements comprises a set of instructions for creating at least one association between the set of primitive graphic elements that are used to define the structural graphic element.

107. The computer readable medium of claim 92, wherein the computer program further comprises a set of instructions for defining a structured document based on the structural graphic element.

108. The computer readable medium of claim 107, wherein the structured document comprises a hierarchical model of the document comprising a plurality of nodes, wherein one of the nodes is populated by the structural graphic element.

109. A method for defining a program for analyzing a document and generating structural elements that define structure in the document based on the analysis, the method comprising:

   defining a module for receiving the document that comprises a plurality of primitive graphic elements, each primitive graphic element defined as a single object in the document, the document having a drawing order that indicates the order in which the primitive graphic elements are drawn when the document is displayed;

   defining a module for identifying positional relationships between successive primitive graphic elements in the drawing order; and

   defining a module for, based on the positional relationships, defining a single structural graphic element from at least two of the primitive graphic elements.

110. The method of claim 109 further comprising defining a module for defining a structured document that includes the structural graphic element.

111. The method of claim 110, wherein the structured document comprises a hierarchical model of the document comprising a plurality of nodes, wherein one of the nodes is populated by the structural graphic element.

112. A computer readable medium storing a computer program which when executed by at least one processor analyzes a document, the computer program comprising sets of instructions for:
receiving the document that comprises a plurality of primitive graphic elements, each primitive graphic element defined as a single object in the document, the document having a drawing order that indicates the order in which the primitive graphic elements are drawn when the document is displayed;

calculating values for each pair of successive primitive graphic elements in the drawing order, wherein the calculated values relate to a size of the primitive graphic elements in the pair;

based on the calculated values, defining a cluster of successive primitive graphic elements; and

identifying a set of sub-clusters of primitive graphic elements in the cluster that satisfy particular constraints; and

defining each particular sub-cluster as a single structural graphic element comprising the primitive graphic elements in the particular sub-cluster.

113. The computer readable medium of claim 112, wherein the calculated values further relate to the proximity to each other of the primitive graphic elements in the pair.

114. A method for defining a program for reconstructing a document, the method comprising:

defining a default set of document reconstruction operations for defining a structured document from a document that comprises a plurality of primitive elements;

defining a hierarchical set of profiles, each profile comprising (i) a set of potential document reconstruction results and (ii) instructions for modifying the document reconstruction operations when intermediate document reconstruction results match the potential document reconstruction results for the profile, wherein instructions from a profile at a lower level in the hierarchy override instructions from a profile at a higher level; and

defining a module for matching intermediate document reconstruction results to a profile.

115. The method of claim 114, wherein the structured document comprises a hierarchical model of the document.

116. The method of claim 114, wherein the hierarchical set of profiles comprises profiles for at least two of documents, sections, pages, and zones.

117. The method of claim 116, wherein each document profile specifies a particular set
of possible page profiles and each page profile specifies a particular set of possible zone profiles.

118. The method of claim 117, wherein the set of page profiles for a first document profile comprises a particular page profile and the set of page profiles for a second document profile also comprises the particular page profile.

119. The method of claim 114, wherein the instructions of a particular profile comprise a subset of profiles at a lower level in the hierarchical set of profiles that should be tested when the intermediate document reconstruction results match the potential document reconstruction results for the particular profile.

120. The method of claim 114, wherein the instructions of a particular profile comprise instructions to not perform a particular document reconstruction operation.

121. The method of claim 120, wherein the particular profile is a profile for a page and the set of potential document reconstruction results for the particular profile comprises results that the page has only one column of text, wherein the instructions for the particular profile comprise instructions to not perform operations for splitting lines of text.

122. The method of claim 114, wherein the instructions of a particular profile comprise instructions regarding how to perform a particular document reconstruction operation.

123. The method of claim 114, wherein the instructions of a particular profile comprise instructions specifying a particular reconstruction process the results of which should be removed from the intermediate reconstruction results.

124. A computer readable medium storing a computer program for execution by at least one processor, the computer program comprising sets of instructions for:
   receiving a document comprising a plurality of primitive elements;
   identifying a default set of document reconstruction operations for reconstructing the document to define a structured document;
   performing one or more of the document reconstruction operations from the default set;
   based on results of the performed document reconstruction operations, identifying a profile for the document; and
   modifying the set of document reconstruction operations for reconstructing the document according to the identified profile.

125. The computer readable medium of claim 124, wherein the computer program
further comprises a set of instructions for performing at least one of the document reconstruction operations from the modified set after modifying the set.

126. The computer readable medium of claim 124, wherein the set of instructions for modifying the set of document reconstruction operations comprises a set of instructions for removing a particular reconstruction operation from the set of reconstruction operations to be performed.

127. The computer readable medium of claim 124, wherein the set of instructions for modifying the set of document reconstruction operations comprises a set of instructions for adding a particular reconstruction operation to the set of reconstruction operations to be performed.

128. The computer readable medium of claim 124, wherein the default set of reconstruction operations specifies an order in which the reconstruction operations are to be performed.

129. The computer readable medium of claim 128, wherein the set of instructions for modifying the set of document reconstruction operations comprises a set of instructions for modifying the order in which the reconstruction operations are to be performed.

130. The computer readable medium of claim 124, wherein the profile is a document profile that identifies a type of document for the document based on results of the performed document reconstruction operations.

131. The computer readable medium of claim 130, wherein the computer program further comprises sets of instructions for:

- performing at least one of the document reconstruction operations from the modified set after modifying the set;

- based on results of the performed document reconstruction operations, identifying a section profile for a particular section of the document; and

- modifying the set of document reconstruction operations for reconstructing the particular section of the document according to the identified section profile.

132. The computer readable medium of claim 131, wherein the particular section is a particular page.

133. The computer readable medium of claim 131, wherein instructions in the section profile override conflicting instructions in the document profile for reconstructing the particular
134. The computer readable medium of claim 131, wherein instructions in the
document profile that are not contradicted by instructions in the section profile are still in
operation after the identification of the section profile.

135. A computer readable medium storing a computer program which when executed
by at least one processor displays a document, the computer program comprising sets of
instructions for:

- defining a structured document comprising a hierarchy of structural elements, the
  hierarchy constructed by analyzing an unstructured document;
- displaying the document on a device;
- receiving a selection of a position of interest in the document;
- based on the position of interest, identifying a structural element within the
  hierarchy as a region of interest; and
- modifying the display of the document to highlight the identified region of
  interest.

136. The computer readable medium of claim 135, wherein the device is a small-
screen device.

137. The computer readable medium of claim 136, wherein the device is a handheld
device.

138. The computer readable medium of claim 136, wherein the unstructured document
comprises a plurality of unassociated primitive elements, wherein the set of instructions for
defining the structured document comprises sets of instructions for:

- creating associations between the primitive elements to define structural elements;

and

- defining the hierarchy of structural elements.

139. The computer readable medium of claim 135, wherein the set of instructions for
identifying the structural element comprises sets of instructions for:

- identifying the structural element at a lowest level of the hierarchy that includes
  the position of interest; and

- identifying structural elements at higher levels of the hierarchy that include the
  structural element identified at the lowest level of the hierarchy until a structural element
qualifying as a region of interest is reached.

140. The computer readable medium of claim 139, wherein the structural element at the lowest level of the hierarchy is a set of glyphs associated as a word, and the structural element qualifying as a region of interest is a paragraph that includes the word.

141. The computer readable medium of claim 139, wherein the position of interest is within a graphic element and the region of interest is the graphic element.

142. The computer readable medium of claim 135, wherein the set of instructions for receiving a selection of a position of interest comprises a set of instructions for receiving a double tap at a particular position on a touchscreen device.

143. The computer readable medium of claim 135, wherein the set of instructions for modifying the display of the document comprises a set of instructions for a combination of zooming, rotating, and translating of the document in order to center the region of interest.

144. The computer readable medium of claim 135, wherein the set of instructions for modifying the display of the document comprises sets of instructions for:

   defining a rectangle of interest based on the structural element that is the region of interest; and

   zooming the display to maximize the size of the rectangle of interest while displaying the entire rectangle of interest.

145. The computer readable medium of claim 144, wherein the rectangle of interest is zoomed such that a length of the rectangle in a first direction is equal to a length of a display on the device in the first direction, wherein the set of instructions for modifying the display of the document further comprises sets of instructions for centering the rectangle of interest in the display in a second direction.

146. The computer readable medium of claim 145, wherein the first direction is horizontal and the second direction is vertical.

147. The computer readable medium of claim 145, wherein the first direction is vertical and the second direction is horizontal.

148. The computer readable medium of claim 135, wherein the computer program further comprises sets of instructions for:

   receiving an input to move from the region of interest; and

   modifying the display of the document to highlight a structurally related region of
149. The computer readable medium of claim 148, wherein the structurally related region of interest is a structural element that is previous to the identified structural element in a reading order.

150. The computer readable medium of claim 148, wherein the structurally related region of interest is a structural element that is after the identified structural element in a reading order.

151. The computer readable medium of claim 148, wherein the structurally related region of interest is a structural element on a different page in the document than the initial region of interest, wherein the structured document includes a link between the region of interest and the structurally related region of interest.

152. A computer readable medium storing a computer program for a handheld display device, the computer program for execution by at least one processor, the computer program comprising sets of instructions for:

analyzing an unstructured document that comprises a plurality of unassociated primitive elements to define a structured document comprising a hierarchy of structural elements, the structural elements defined by associating sets of primitive elements;

displaying, on the handheld display device, a particular portion of the structured document in a first layout when the display device is at a first orientation;

displaying, on the handheld display device, the particular portion of the structured document in a second layout different than the first when the display device is at a second orientation.

153. The computer readable medium of claim 152, wherein the first orientation is a horizontal orientation and the second orientation is a vertical orientation.

154. The computer readable medium of claim 153, wherein the portion of the structured document comprises text that is arranged in one column, wherein the first layout displays the text in three columns and the second layout displays the text in two columns.

155. The computer readable medium of claim 152, wherein the handheld display device comprises an accelerometer for determining the orientation of the display device.

156. The computer readable medium of claim 152, wherein the handheld display device is an iPhone®.
157. A method for defining a program for displaying a document, the method comprising:

- defining a module for defining a structured document comprising a hierarchy of structural elements, the hierarchy constructed by analyzing an unstructured document;
- defining a module for displaying the document on a device;
- defining a module for receiving a selection of a position of interest in the document;
- defining a module for identifying, based on the position of interest, a structural element within the hierarchy as a region of interest; and
- defining a module for modifying the display of the document to highlight the identified region of interest.

158. The method of claim 157, wherein the program is defined for firmware of a handheld device.

159. A computer readable medium storing a computer program which when executed by at least one processor defines a selection of text in a document, the computer program comprising sets of instructions for:

- receiving an unstructured document comprising a plurality of unassociated glyphs;
- associating sets of glyphs;
- identifying a reading order that specifies a flow of reading through the glyphs;
- displaying the document;
- receiving a start point and end point for a selection of text within the displayed document; and
- defining a selection of text from the start point to the end point by using the identified sets of glyphs and intended flow of reading.

160. The computer readable medium of claim 159, wherein the associated sets of glyphs comprise a plurality of paragraphs and the reading order specifies a flow of reading from a first paragraph to a second paragraph that is not contiguous.

161. The computer readable medium of claim 159, wherein the associated sets of glyphs comprise a plurality of columns, and the start point and end point are in different columns.
162. The computer readable medium of claim 161, wherein the reading order specifics an order value for each column, wherein the selection of text includes all columns with order values in between the order value of the column including the start point and the order value of the column including the end point.

163. The computer readable medium of claim 159, wherein the computer program further comprises sets of instructions for:
   determining whether the start point is after the end point in the reading order;
   when the start point is after the end point, switching the start point and end point for the selection so that the start point is before the end point in the reading order.

164. The computer readable medium of claim 159, wherein the start point and end point for the text selection are received through a click and drag selection with a cursor controller.

165. The computer readable medium of claim 159, wherein the start point is a location of a cursor when a cursor controller button is pressed and the end point is a location of the cursor after the cursor is moved from the start point.

166. The computer readable medium of claim 159, wherein the start point and end point for the text selection are received through a touchscreen gesture.

167. The computer readable medium of claim 166, wherein the start point is a location at which a user initially touches the touchscreen with an object and the end point is a location to which the user has dragged the object.

168. The computer readable medium of claim 167, wherein the object is the user's finger.

169. The computer readable medium of claim 159, wherein the set of instructions for identifying associated sets of glyphs and a reading order comprises a set of instructions for defining a structured document from the unstructured document, wherein the associated sets of glyphs are structural elements in the structured document.

170. The computer readable medium of claim 169, wherein the structured document includes a plurality of layouts, each layout comprising one or more associated sets of glyphs.

171. The computer readable medium of claim 170, wherein the computer program further comprises sets of instructions for:
   determining whether the start point and end point are within a same layout; and
when the start point and end point are not in the same layout, defining a new end
point that is in the same layout as the start point.

172. The computer readable medium of claim 171, wherein the set of instructions for
defining a new end point comprises;
identifying a straight line from the start point to the end point; and
defining a point at which the line leaves the layout of the start point as the new
end point.

173. The computer readable medium of claim 170, wherein the set of instructions for
defining a selection of text comprises sets of instructions for:
determining whether the start point and end point are in the same layout; and
when the start point and end point are in different layouts, defining the selection
to include all text in the layout of the start point that is after the start point in the layout.

174. The computer readable medium of claim 173, wherein the set of instructions for
defining a selection of text further comprises a set of instructions for defining the selection to
include all text in any layouts between the layout of the start point and the layout of the end
point.

175. The computer readable medium of claim 159, wherein the selection of text is for
copying and pasting into a second document, wherein the text is pasted into the second document
with the reading order maintained.

176. A computer readable medium storing a computer program for execution by at
least one processor, the computer program comprising sets of instructions for:
analyzing an unstructured document that comprises a plurality of primitive
elements to define a structured document as a hierarchical model of the unstructured document,
the structured document comprising a plurality of structural elements and a reading order
through the structural elements;
displaying the document;
receiving a start point and end point for a selection of text in the document; and
selecting text in the document from the start point to the end point by using the
structural elements and reading order through the structural elements.

177. The computer readable medium of claim 176, wherein the document is displayed
on a handheld display device.
178. The computer readable medium of claim 176, wherein the start point is in a column of text within an article and the end point is within a call-out for the article.

179. The computer readable medium of claim 178, wherein the computer program further comprises a set of instructions for moving the end point to a column of text in the article.

180. The computer readable medium of claim 178, wherein the set of instructions for selecting text comprises a set of instructions for selecting all text in the columns of the article.

181. A method for defining a program for defining a selection of text in a document, the method comprising:

  - defining a module for receiving an unstructured document comprising a plurality of unassociated glyphs;
  - defining a module for associating sets of glyphs;
  - defining a module for identifying a reading order that specifies a flow of reading through the glyphs;
  - defining a module for displaying the document;
  - defining a module for receiving a start point and end point for a selection of text within the displayed document; and
  - defining a module for defining a selection of text from the start point to the end point by using the identified sets of glyphs and intended flow of reading.

182. A computer readable medium storing a computer program which when executed by at least one processor defines structure for a document comprising a plurality of primitive elements that are defined in terms of their position in the document, the computer program comprising sets of instructions for:

  - identifying, for a particular set of primitive elements, distances between pairs of nearest primitive elements;
  - sorting the identified distances;
  - storing a single value that identifies which of the identified distances are greater than a given threshold for a partition; and
  - using the stored single value to identify and analyze the partition in order to define structural elements for the document.

183. The computer readable medium of claim 182, wherein the given threshold for the partition identifies a minimum distance between a pair of primitive elements.
184. The computer readable medium of claim 183, wherein pairs of primitive elements that are closer than the minimum distance are in a same cluster of primitive elements.

185. A computer readable medium storing a computer program which when executed by at least one processor defines structure for a document comprising a plurality of primitive elements that are defined in terms of their position in the document, the computer program comprising sets of instructions for:

- identifying, for a particular set of primitive elements, a pairwise grouping of nearest primitive elements in the set;
- sorting the pairwise groupings of primitive elements based on an order from the closest to the furthest pairs;
- storing a single value that identifies which of the pairwise groupings of primitive elements are sufficiently far apart to form a partition; and
- using the stored single value to identify and analyze the partition in order to define structural elements for the document.

186. The computer readable medium of claim 185, wherein the computer readable medium is firmware of a handheld device.

187. The computer readable medium of claim 186, wherein the device is a cell phone.

188. The computer readable medium of claim 186, wherein the device is a media player.

189. The computer readable medium of claim 185, wherein the computer program further comprises a set of instructions for defining a structured document that comprises the structural elements and the primitive elements.

190. The computer readable medium of claim 185, wherein:

- the document comprises a plurality of glyphs;
- the computer program further comprises a set of instructions for associating sets of glyphs as words;
- the particular set of primitive elements are leftmost glyphs of words; and
- the set of instructions for using the stored single value to identify and analyze the partition in order to define structural elements comprises a set of instructions for defining left alignment guides for sets of aligned words.

191. The computer readable medium of claim 185, wherein:
the document comprises a plurality of glyphs;
the computer program further comprises a set of instructions for associating sets of glyphs as words;
the particular set of primitive elements are rightmost glyphs of words; and
the set of instructions for using the stored single value to identify and analyze the partition in order to define structural elements comprises a set of instructions for defining right alignment guides for sets of aligned words.

192. The computer readable medium of claim 185, wherein the particular set of primitive elements are primitive graphic elements and the set of instructions for using the stored single value to identify and analyze the partition in order to define structural elements comprises a set of instructions for associating nearby primitive graphic elements as structural graphic elements.

193. The computer readable medium of claim 185, wherein the primitive elements are glyphs, and the set of instructions for defining structural elements comprises a set of instructions for associating sets of nearby glyphs as words.

194. The computer readable medium of claim 193, wherein the set of instructions for identifying a pairwise grouping of nearest primitive elements comprises sets of instructions for:
identifying a set of glyphs with a common baseline;
sorting the set of glyphs based on coordinates of the glyphs in a first direction;
for each pair of consecutive glyphs in the sorted set, calculating a difference in coordinate values between the glyphs in the pair; and
storing the calculated differences in a first array.

195. The computer readable medium of claim 194, wherein the set of instructions for sorting the pairwise primitive elements based on an order from the closest to the furthest pairs comprises sets of instructions for:
sorting the calculated difference values; and
storing, in a second array, the indices of the first array that correspond to the sorted differences.

196. The computer readable medium of claim 195, wherein the index of the first array that stores a smallest difference value is stored as the value at the first index of the second array.

197. The computer readable medium of claim 195, wherein the computer program
Further comprises a set of instructions for determining a minimum difference between glyphs, wherein the stored single value is an index in the second array at which the index of the first array storing said minimum difference is stored.

198. The computer readable medium of claim 197, wherein the minimum difference represents gaps between words.

199. The computer readable medium of claim 197, wherein the set of instructions for using the stored value to identify and analyze the partitions comprises a set of instructions for splitting a third array that stores the sorted x-coordinate values at indices of the third array that are stored as values in the second array at all indices of the second array at and after the index stored as the single value.

200. The computer readable medium of claim 185, wherein the set of instructions for storing the single value comprises sets of instructions for storing a plurality of single values that each identify different partitions for the primitive elements.

201. The computer readable medium of claim 200, wherein the computer program further comprises analyzing the different partitions for the primitive elements in order to identify an ideal optimal partition representing an optimal distance scale.

202. A method for defining a program for defining structure for a document, the method comprising:

   defining a module for identifying a pairwise grouping of nearest primitive elements in a document comprising a plurality of primitive elements that are defined in terms of their position in the document;

   defining a module for sorting the pairwise groupings of primitive elements based on an order from the closest to the furthest pairs;

   defining a module for storing a single value that identifies which of the pairwise-grouped primitive elements are sufficiently far apart to form a partition; and

   defining a module for using the stored value to identify and analyze the partitions in order to define structural elements for the document.

203. The method of claim 202 further comprising defining a module for defining a structured document based on the structural elements and the primitive elements.

204. The method of claim 203, wherein the structured document is a hierarchical structure in which the structural elements are nodes.
205. A method comprising:

- defining a plurality of different processes for analyzing and manipulating a document comprising a plurality of primitive elements; and
- defining a storage for data associated with the primitive elements, wherein at least some of the data is stored in a separate memory space from the processes and is shared by at least two different processes, wherein the processes access the data by use of references to the data, wherein the data is not replicated by the processes.

206. The method of claim 205, wherein the storage for data associated with the primitive elements is a first storage, the method further comprising defining a parser for (i) parsing the document to identify the primitive elements and (ii) storing the primitive elements in a second storage.

207. The method of claim 206, wherein the data associated with the primitive elements comprises references to the primitive elements in the second storage.

208. The method of claim 206, wherein the parser is for (i) parsing the document to identify the primitive elements in a random order and (ii) storing the primitive elements in the random order in the second storage.

209. The method of claim 206, wherein the random order is the order in which the primitive elements are defined in a bitstream representation of the document.

210. The method of claim 206 further comprising defining a module for (i) sorting the primitive elements according to a particular metric and (ii) storing the data associated with the primitive elements in the first storage in a sorted order.

211. The method of claim 210, wherein the primitive elements comprise information regarding their location in the document, wherein the particular metric is based on the location of the primitive elements.

212. The method of claim 210, wherein within a particular page of the document, the particular metric sorts the primitive elements top to bottom as a primary metric and left to right as a secondary metric.

213. The method of claim 205, wherein the primitive elements comprise a plurality of glyphs.

214. The method of claim 213, wherein the plurality of different processes comprises:
- a process for associating sets of glyphs as lines of text; and

175
a process for associating sets of glyphs as words.

215. The method of claim 214, wherein:
the process for associating sets of glyphs as lines of text stores a particular line of
text as a first string that references the data associated with the primitive elements; and
the process for associating sets of glyphs as words stores a particular word as a
second string that references the same data associated with the primitive elements.

216. The method of claim 215, wherein the same data is for use by both processes
without replication.

217. The method of claim 215, wherein each of the first and second strings reference
the data by storing only two values.

218. The method of claim 217, wherein the two values are a reference to a first piece
of data and a count of the number of pieces of data in the string.

219. The method of claim 217, wherein the first string and second string reference the
same first piece of data and store different counts.

220. The method of claim 205 further comprising defining a shared memory" data
structure for managing the storage for data associated with the primitive elements.

221. The method of claim 220, wherein the shared memory data structure is for storing
a location in the separate memory space at which the data associated with the primitive elements
is stored.

222. The method of claim 220, wherein the shared memory data structure is for use
when at least two other data structures reference the data associated with the primitive elements.

223. The method of claim 222, wherein the at least two other data structures share
ownership of the data associated with the primitive elements.

224. The method of claim 220, wherein the shared memory data structure is for storing
the number of other data structures that reference the data associated with the primitive elements.

225. A computer readable medium storing a computer program for execution by at
least one processor, the computer program comprising sets of instructions for:
parsing a document comprising a plurality of primitive elements;
storing the primitive elements in a random order in a first storage;
storing references to the primitive elements in a second storage in an order based
on locations of the primitive elements in the document;
receiving instructions to perform a document reconstruction operation; and
performing the received instructions without storing any new references to the primitive elements.

226. The computer readable medium of claim 225, wherein the computer program further comprises sets of instructions for:

receiving a second set of instructions to perform a second document reconstruction operation;
determining that performing the second document reconstruction operation requires new references to the primitive elements; and
storing new references to the primitive elements in a third storage in an order different from that of the second storage.

227. The computer readable medium of claim 226, wherein the second storage and third storage are in the same physical storage.

228. A method comprising:
defining a first module for (i) parsing a document comprising a plurality of primitive elements and (ii) storing the primitive elements in a random order in a first storage;
defining a second module for (i) allocating memory in a second storage for storing references to the randomly-ordered primitive elements and (ii) storing the references in a particular order in the allocated memory;
defining a third module for storing a data structure that references a portion of the ordered references, the data structure comprising only a reference to a first one of the ordered references and a count value; and
defining a fourth module for (i) receiving instructions to perform document reconstruction operations and (ii) identifying which of the first, second, and third modules is required to perform the document reconstruction operations while minimizing memory and computation usage.

229. The method of claim 228, wherein when only the third module is necessary to perform a particular document reconstruction operation, no new memory is allocated for either copies of the primitive elements or new references to the primitive elements.

230. The computer readable medium of one of claims 1, 22, 30, 39, 44, 46, 50, 61, 67, 71, 88, 92, 112, 124, 182, 185, and 225, wherein the document is an unstructured document.
231. The computer readable medium of one of claims 1, 22, 30, 39, 44, 46, 50, 61, 67, 71, 88, 92, 112, 124, 182, 185, and 225, wherein the document is a vector graphics document.

232. The computer readable medium of one of claims 1, 22, 30, 39, 44, 46, 50, 61, 67, 71, 88, 92, 112, 124, 182, 185, and 225, wherein the document is a Portable Document Format (PDF) document.

233. The computer readable medium of one of claims 1, 22, 30, 39, 44, 46, 50, 61, 67, 71, 88, 92, 112, 124, 135, 152, 159, 176, 182, 185, and 225, wherein the computer program is an application for a handheld device.

234. The computer readable medium of one of claims 1, 22, 30, 39, 44, 46, 50, 61, 67, 71, 88, 92, 112, 124, 135, 152, 159, 176, 182, 185, and 225, wherein the computer program is a portion of an operating system.

235. The computer readable medium of one of claims 1, 22, 30, 39, 44, 46, 50, 61, 67, 71, 88, 92, 112, 124, 135, 152, 159, 176, 182, 185, and 225, wherein the computer program is an application that runs on top of an operating system.

236. The computer readable medium of one of claims 135, 152, 159, and 176, wherein the unstructured document is a vector graphics document.

237. The computer readable medium of one of claims 135, 152, 159, and 176, wherein the unstructured document is a portable document format (PDF) document.
(3/111)

**Figure 3**

- Start
- Receive page
- Identify zones on page
- Generate zone tree for page
- Insert content of page into zone tree
- End
Figure 6
Figure 7

Figure 8
(8/111)

Start

1. Receive rotation group and normalize to upright
2. Identify potential zone borders
3. Remove borders or portions of borders that intersect with other objects
4. Merge borders

- Any remaining merged borders?
  - No
    - Identify zone border intersections
  - Yes
    - Merged border too thick or includes too many zone border graphics?
      - Yes
        - Remove as border and classify as compound graphic
      - No

5. Eliminate borders with less than two intersections
6. Trim zone borders
7. Store border and intersection information

End

Figure 9
Figure 10
Figure 12

1200
Receive zone borders and intersections

1207
Any zone border intervals?
Yes
Assign two direction vectors to each zone border interval

1210
No
Select a border interval \( b \), an intersection \( i \), and a direction \( d \)

1215
Proceed in direction \( d \) from intersection \( i \) until arriving at next intersection

1220
Set direction \( d \) moving away from intersection \( i \) along border \( b \)

1225
Intersection is original intersection?
Yes
Define zone \( Z \) as set of border intervals traversed

1265
No
Remove all direction vectors used in traversal of \( Z \)

1270
Remove all border intervals with no remaining direction vectors

1275
When \( Z \) is island, classify as such

1280
Yes
Zone border intervals remaining?
No
Store zone information
End

1290
1230
Can turn clockwise at intersection?
Yes
Turn clockwise to next border interval

1235
No
Continue straight to next border interval

1240
Can continue straight through intersection?
Yes
Set new border interval as current border interval \( b \), new intersection as intersection \( i \)

1245
No
Turn counterclockwise to next border interval

1250
1255
1250
1255
Any unprocessed content objects?

Yes

Select content object c

Define node for c

Determine smallest zone \( x \) that contains content object c

Define edge from \( x \) to c

Store zone graph

End

Figure 15B
Figure 16
Figure 17
Start

1805

Receive portion of document

1810

Apply cluster analysis to determine guides of document portion

1815

Determine gutters of document portion

1820

Use guides and gutters to semantically reconstruct document

End

Figure 18
Start

Receive set of input data

Sort set of input data

Set has at least two data?

Yes

Determine set of differences between subsequent data in sorted set

Set $d$ to largest unevaluated difference in set of differences

No

Partition sorted data wherever difference is greater than or equal to $d$ to generate set of subsets

Determine set $S$ of subsets that satisfy constraints

$S$ is empty set?

Yes

Evaluate optimization function for $S$

$S$ most optimal so far?

No

Any unevaluated differences?

Yes

Output currently stored best set as set of clusters

End

No

Store $S$ as best set of clusters

Figure 20
Start

Set input data as x-coordinates of left edge of words in region

Determine cluster properties

Apply cluster analysis to input data to find clusters of x-coordinate values

Any unevaluated clusters?

No

End

Yes

Select cluster

Set left-align guide as rectangle with minimum and maximum x-coordinates of cluster and minimum and maximum y-coordinates of top and bottom of page

Erase guide at y-coordinates that do not satisfy constraints based on analysis of words starting in rectangle and crossing rectangle
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis minim veniam, quis nostrud irure dolor in reprehenderit in voluptate.

Nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Figure 22
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in.

Nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in.

Culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in.

Figure 23
Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate.

Nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Exceper sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. 

Velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
Figure 25

Start

Set input data as x-coordinates of right edge of words in region

Determine cluster properties

Apply cluster analysis to input data to find clusters

Any unevaluated clusters?

Yes

Select cluster

Set right-align guide as rectangle with minimum and maximum x-coordinates of cluster and minimum and maximum y-coordinates of top and bottom of page

Erase guide at y-coordinates that do not satisfy constraints based on analysis of words ending in rectangle and crossing rectangle

End

Figure 25
Start

Receive alignment information for region

Any unprocessed right-alignment points?

Yes

Select right-alignment point

No

Left-alignment point between right-alignment point and region edge?

Yes

Moving right across region, identify next left-alignment point

Set gutter as rectangle with minimum x-coordinate as left-alignment point, maximum x-coordinate as right-alignment point, and minimum and maximum y-coordinates as top and bottom of page

Erase gutter at y-coordinates that do not satisfy constraints based on analysis of words crossing into rectangle

End

Figure 26
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate.

Nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, cons

Velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, cons

Culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea.

Figure 27
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate.

Nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna.

Velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea.

Figure 29
Figure 30
Start

3105 Receive portion of document

3110 Identify lines of text

3115 Identify words in text

3120 Split lines of text where the text is discontinuous

3125 Place lines into paragraphs

3130 Place paragraphs into columns and layouts

End

Figure 31
Figure 36

Start

Receive document portion

Identify locations of elements

Determine first-order differences between locations of adjacent elements

Sort first-order differences

Determine second-order differences between sorted first-order differences

Determine number of cluster levels by analyzing second-order differences

Determine level characteristics

End
Figure 38A

Start

Receive text lines, guide and gutter information, and segment break information for document portion

Sort text lines based on y-coordinate of baseline

Select bottom-most unevaluated text line

Identify potential splits in selected line

Any unevaluated potential splits?

Yes

Select a potential split from current text line

X-interval of potential split overlaps with x-interval of any potential split from previous line?

Yes

Associate overlapping potential splits

No

More unevaluated lines?

Yes

No

to 3847
Start

Receive paragraphs for portion of document

Any paragraphs?

No

Yes

Select paragraph

Calculate in-order, out-order, left-order, right-order, and accompanying sets for selected paragraph

More paragraphs?

No

Generate flow graph for paragraphs

Identify call-outs and remove call-outs from consideration

Merge nodes of flow graph into columns

Partition flow graph into layouts

End

Figure 42
Figure 43
Figure 47
Figure 48

Start

Receive zone and layout information

Identify tables with complete borders

Identify tables with connected borders

Identify tables with no borders or disconnected borders

Define tables to be accessible by editing and other software

End
(50/111)

**Figure 49**

<table>
<thead>
<tr>
<th>Column Heading</th>
<th>Subhead 1</th>
<th>Subhead 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row A</td>
<td>content</td>
<td>content</td>
</tr>
<tr>
<td>Row B</td>
<td>content</td>
<td>content</td>
</tr>
<tr>
<td>Row C</td>
<td>content</td>
<td>content</td>
</tr>
</tbody>
</table>

**Figure 50**

<table>
<thead>
<tr>
<th>Sport</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Soccer</td>
<td>competitive</td>
</tr>
<tr>
<td></td>
<td>team</td>
</tr>
<tr>
<td>Chess</td>
<td>competitive</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Yoga</td>
<td>introspective</td>
</tr>
<tr>
<td></td>
<td>solo</td>
</tr>
<tr>
<td></td>
<td>active</td>
</tr>
<tr>
<td>Soccer</td>
<td>Chess</td>
</tr>
<tr>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>competitive</td>
<td>competitive</td>
</tr>
<tr>
<td>team</td>
<td>Soccer</td>
</tr>
<tr>
<td>active</td>
<td>inactive</td>
</tr>
</tbody>
</table>

**Figure 51**

5100

<table>
<thead>
<tr>
<th>Soccer</th>
<th>Chess</th>
<th>Yoga</th>
</tr>
</thead>
<tbody>
<tr>
<td>competitive</td>
<td>competitive</td>
<td>introspective</td>
</tr>
<tr>
<td>team</td>
<td>Soccer</td>
<td>solo</td>
</tr>
<tr>
<td>active</td>
<td>inactive</td>
<td>active</td>
</tr>
</tbody>
</table>

**Figure 52**
Figure 53A

Start

Receive zone information for portion of document

Identify islands partitioned entirely by two or more zones, all of which are rectangular

Any unprocessed islands?

Select next island

Sort horizontal borders in selected island from highest to lowest

Select first unevaluated horizontal border

Define next horizontal gridline of table

Assign border to current gridline and set vertical extent of gridline to intersection of y-intervals of borders assigned to gridline

Y-interval of Next border overlaps current gridline?

More horizontal borders?

Sort vertical borders in selected island from left to right

Select first unevaluated vertical border

Define next vertical gridline of table

End
Figure 53B
Start

Receive set of potential zone borders and border intersections for document portion

Define set U to include all potential zone borders

U is empty?

Yes

No

Select border b from U and define new connected set C

Remove b from U, place b in C

Select border z from U that has not been evaluated for current b

z intersects b?

Yes

Add z to C and place z in queue

No

Queue is empty?

Yes

No

Any borders in U not evaluated for intersection with b?

No

Yes

Set next border in queue as new b

U is empty?

Yes

No

to 5565
to end

Figure 55A
Figure 55B

1. From 5560:
   - Select a connected set C

2. From 5560:
   - Add to C the four borders that form the upright bounding box of C

3. Branch with condition:
   - Any C remaining?
     - Yes:
     - No:
       - Identify zones using all borders from all connected sets
       - Identify tables among zones and reconstruct tables

4. End
Define implied zone borders at vertical white space separating columns of layout and horizontal white space separating paragraphs of layout.

Any horizontal border in layout which, extended outward, intersects bounding box of paragraph in another column of layout?

Add layout to set S.

Apply zone analysis to implied zone borders from layouts of S.

Identify potential tables among layouts of S.

Disqualify any potential tables that fail to meet all table conditions.

End.

Figure 57B
Figure 58
Figure 60
Figure 61
Figure 62
Start

Receive graphs of document portion

At least two graphs?

No

Yes

Sort graphs by drawing order

Set first graph in drawing order as current graph g

g is last graph?

No

Calculate spread between g and next graph in drawing order

Store spread in array

Set next graph as current graph g

Use spreads as 1st-order differences for difference clustering to define clusters of graphs

Select current cluster C from unevaluated clusters

Process C into set of subsequences meeting constraints

More clusters?

Yes

No

End

Figure 64
Figure 67
Figure 68

Figure 69
(72/111)

Start

1. Receive content to be processed
2. Select profile for top level in hierarchy
3. Apply processing for selected profile

4. Processing far enough along to proceed to the next level of hierarchy?
   - No
   - Yes
     a. Select profile for next level of hierarchy
     b. Modify methods from previous levels of hierarchy based on newly selected profile
     c. Apply processing methods for selected profiles

5. Any lower levels of hierarchy?
   - No
   - Yes
     a. Processing complete?
       - No
       - Yes
         End
     - No
       Processing far enough along for next level of hierarchy?
       - No
       - Yes

Figure 71
Start

Receive document

Identify profile for document

Identify methods associated with identified profile

Select one of identified methods

Selected method explicitly defined for current profile?

Select implementation of method for current profile

Selected method defined for active profile at higher level of hierarchy?

Select implementation of method from higher level active profile

Select default implementation of method

Perform reconstruction using selected implementation

New profile identified?

No

Reconstruction complete?

Yes

End

No
Figure 73
Figure 78
Start

Receive position of interest

Determine selected object in semantically reconstructed document based on received position of interest

Move up hierarchy of document until paragraph or graphic object is identified

Identified object is paragraph?

Yes

Define rectangle with width of parent column of paragraph and height of paragraph

Apply any rotation for paragraph to rectangle and define ROI as upright bounding box of rotated rectangle

No

Zoom such that width of ROI is no larger than width of display area

Paragraph baselines steeper than diagonal of display area?

Yes

Zoom such that ROI is no larger than height of display area

Zoom such that width and height of ROI are no larger than width and height of display area

No

ROI fits in display area horizontally?

Yes

Center ROI horizontally

No

Center position of interest horizontally

ROI is upright bounding box of identified object

Figure 80A
from 8070

8075 ROI fits in display area vertically?

Yes → Center ROI vertically

No → Center position of interest vertically

8080

8085

8087 Bottom or top edge of page in display area?

No → Center position of interest vertically

Yes → Move visible page edge to corresponding edge of display area

8090

8092 Right or left edge of page in display area?

No → Center position of interest vertically

Yes → Move visible page edge to corresponding edge of display area

8095

8097 Redraw display area with all transformations applied

End

Figure 80B
Start

8105
Receive position of interest

8110
Determine selected object in semantically reconstructed document based on received position of interest

8115
Move up hierarchy of document until paragraph or graphic object is identified

8120
Identified object is paragraph?

8125
ROI is upright bounding box of identified object

8135
Define rectangle with width of parent column of paragraph and height of paragraph

8140
Apply any rotation for paragraph to rectangle

8145
Define ROI as upright bounding box including first particular number of lines above and second particular number of lines below position of interest

8150
Zoom such that width and height of ROI are no larger than width and height of display area

8160
ROI fits in display area horizontally?

8165
Center ROI horizontally

8170
Center position of interest horizontally

to 8175

Figure 81A
from 8170

8175

ROI fits in display area vertically?  Yes

8180

Center ROI vertically

8185

No

Center position of interest vertically

8187

Bottom or top edge of page in display area?  No

8189

Yes

Move visible page edge to corresponding edge of display area

8190

8192

Right or left edge of page in display area?  No

8194

Yes

Move visible page edge to corresponding edge of display area

8195

8197

Redraw display area with all transformations applied

End

Figure 81B
**Figure 84**

1. Start
2. Display region of document
3. Receive input scrolling past end of displayed region
4. Automatically move next region in flow into display using semantically reconstructed hierarchical model of document
5. End

**Figure 82**

1. Start
2. Receive semantically reconstructed document
3. Divide semantically reconstructed document into sections
4. Adapt display of reconstructed document to accommodate natural flow through sections
5. End
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Figure 83
Start

8500

8505 Receive semantically reconstructed document with particular layout

8510 Modify layout for viewing on small-screen device while maintaining flow of document by using hierarchical model of document

8515 Display at least a portion of modified layout on small-screen device

End

Figure 85
Figure 86
(89/111)

Figure 87

Start

8705 Receive zone and layout information for document

8710 Display portion of document

8715 Receive start point of selection

8720 Receive end point of selection

8725 Determine selected section of document using layout and flow properties

End
Figure 89
Start

9405
Receive zone and layout information for document

9410
Assign reading order to each column on each page of document

9415
Display portion of document

9420
Receive start and end point of selection

9425
Define line between start and end point

9430
If start or end point not in a column, define new start or end point at edge of last column line passes through

9435
Start point before end point in reading order?

Yes
Select text from start point to end of column that includes start point, from start of column that includes end point to end point, and all columns in between in reading order

No
Switch start and end point of selection

End

Figure 94
Figure 97
Figure 98
(101/111)

Figure 99

1. Start

2. Receive sorted array \( A \) with data values to be clustered

3. Define and store array \( D(A) \) of first-order differences of \( A \) by comparing pairs of subsequent values of array \( A \)

4. Define and store indirectly sorted array \( S(D(A)) \) of indices of \( D(A) \) by applying sort function to array \( D(A) \)

5. Determine minimum size of gaps between clusters

6. Partition data into clusters using consecutive indices stored in array \( S(D(A)) \)

7. Store partition

8. End
Figure 100
Start

Define indirectly sorted array of differences of values to be clustered

Partition data at several different distance scales concurrently

Store each partition as an integer value referring to an index of indirectly sorted array

Determine optimal distance scale(s)

Store partition of data derived from optimal distance scale as the set of clusters

End

Figure 101
Figure 104
Start

Define geometric analysis modules

Define document reconstruction modules

Define set of hierarchical profiles

Define set of modules for performing cluster analysis

Define modules for adaptively displaying a document

Define modules for receiving user interactions

Define other modules

Store application on a computer readable storage medium

End

Figure 108