Finding Hyper-Structure in Space: Spatial Parsing in 3D

Michael Bang Nielsen & Peter Ørbæk
Dept. of Comp. Sci., University of Aarhus, Aabogade 34, 8200 Aarhus N, Denmark
{bang,poe}@daimi.au.dk

Abstract

Spatial parsers augment spatial hypermedia systems by letting the computer perceive the informal – but visually apparent – groupings formed by humans working with a spatial hypermedia tool. A number of research systems implementing 2D spatial parsing have been described in recent years. This paper extends spatial parsing to 3D and describes an implementation of a tailorable 3D spatial parser for the Topos system: a 3D information organization tool for use on desktops, interactive whiteboards and tables. The parser maintains a proximity graph of the heterogeneous 3D objects and applies structure experts and global repression and reinforcement techniques to this graph to find structures. A number of issues pertaining to 3D parsing as opposed to 2D parsing are discussed. The paper also presents a simple and efficient 2D parser for 3D scenes and compares it to the true 3D parser.

1 Introduction

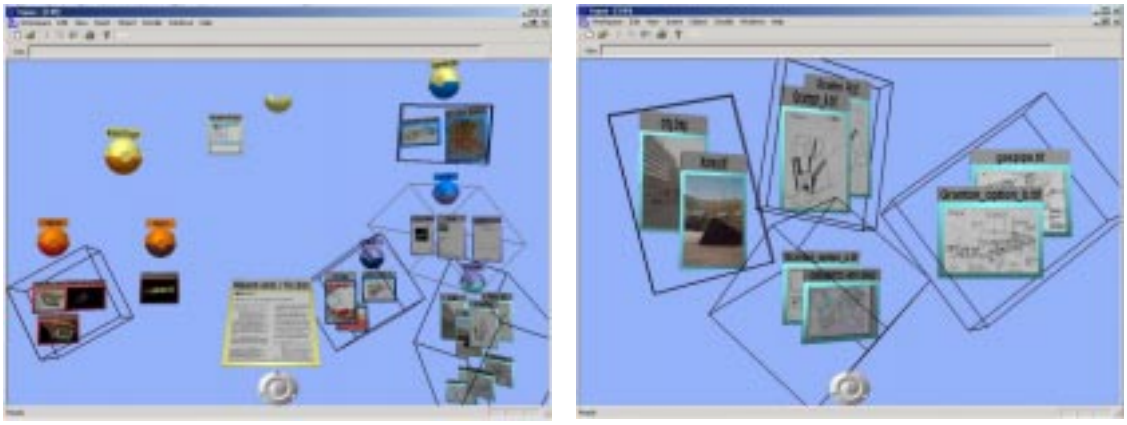
Traditional hypertext considers text and multimedia linked by explicit links. Spatial hypertext considers the process of creating such links by mere placement of documents and materials close to each other on a canvas. Spatial hypertext thereby supports informal, ad hoc, organization of electronic materials. Within the last decade the virtues of spatial hypertext for informal information organization have been explored in the Aquanet (1, 2) and VIKI (3) systems, among others.

Spatial parsing augments spatial hypertext by letting the computer automatically recognize hierarchical spatial groupings on the canvas using the placement (created by users) of work materials for group detection. With help from the parser, such groups can be manipulated as a whole and the visual structure may subsequently be made permanent if the user so desires.

This paper extends the notion of spatial parsing from 2D to 3D and describes the implementation of a 3D spatial parser for the Topos system (4, 5, 6). Topos is a complete re-write of the Manufaktur prototype described in earlier papers. Topos is a prototype of a 3D information management system that supports informal groupings of related documents and 3D models. It has been designed for use on desktops as well as for interactive whiteboards and tables. Topos has been designed from the outset to alleviate the burdens of finite 2D screen space via the use of 3D. The 3D interface of Topos provides a very natural and physical interface metaphor (7) by exploiting users' intuition about 3D space.

Our paper discusses how to implement a spatial parser for 3D spaces. Several questions arise when considering spatial parsing in 3D: do we parse the space in true 3D or just the 2D projection of the space? Does the parse depend on the user's current viewpoint? Also, a number of more technical questions arise, such as how to implement the parser efficiently in the face of true three dimensional objects oriented at arbitrary angles as opposed to the 2D parsers that dealt with groups of axis-aligned rectangles only.

The paper is structured as follows: we first discuss related work; we then briefly describe the Topos system, and describe our approach to spatial parsing in three dimensions. The Implementation section goes into more detail of our spatial parser. We then present an alternative, very simple 2D parser for 3D scenes and compare it to our real 3D parser. The final sections of the paper discuss various aspects of the spatial parser and conclude.



(a) (b)

Figure 1: The same Topos workspace seen from two different viewpoints with the structures found by the parser indicated by wire-frame bounding boxes.

2 Related Work

Spatial parsing was first introduced by Marshall and Shipman in (8), where an implementation of a spatial parser in Aquanet was described. A more mature spatial parser inside the VIKI system was described in (3). Other systems have explored different aspects of spatial parsers: CAOS (9) implemented an incremental spatial parser for a generic hypermedia middle-ware layer (10) intended for distributed collaborative work.

Igarashi, Matsuoka and Masui (11) implemented a standalone adaptive spatial parser using genetic algorithms to tune the parser to the user's expectations. Much of the work in this paper is inspired by their approach and extend it to 3D. The differences between their work and ours will be explained along the way in the following sections, but the primary differences are that they work with axis-aligned rectangles in 2D whereas we work with objects oriented arbitrarily in 3D, they use a local edge selection strategy whereas ours is global and uses constraints based on conflicting structures, and they utilize genetic algorithms to tune parameters whereas we currently rely on user configuration. Additionally our parser is designed as a generic framework which supports structure experts to be plugged in.

The above systems all focus on spatial parsing in a two dimensional canvas. However, it has been repeatedly noted that for large document collections, the limitations of 2D screen space is problematic when organizing large amounts of information (12). Shipman, Marshall and LeMere (13) added multiple fish-eye views to VIKI to try to overcome the lack of 2D screen space, and also support hierarchical spaces that can be individually closed to unclutter the screen.

Several 3D environments for document and task management have been developed in recent years. The WebBook and the Web Forager (14) used a book metaphor in a 3D environment to help organize and provide access to web pages efficiently and intuitively. Data Mountain (15) is another 3D environment for organizing web pages. Data Mountain explored how to exploit human spatial memory (ie. the ability to remember where you put something) in organizing a large number of web pages and verified that users were able to use this ability in a 3D virtual environment. The Task Gallery (16) is a 3D window manager for organizing tasks (a task being a collection of documents and applications) that exploits human spatial memory to keep track of a large number of tasks at once.

The above mentioned 3D environments all take advantage of the intuitive graphical scaling and depth cues that the use of 3D graphics provides, and they verify that by using 3D environments the amount of information presented on the screen can be increased without increasing the cognitive load of the users.

3 Topos and Parsing in 3D

Our 3D parser is implemented as a component inside the Topos system (4, 5, 6). Topos allows for manipulation and maintenance of relationships among materials in a 3D environment. It integrates with existing applications, supports collaboration among co-workers across the Internet and runs on Windows 2000, SGI IRIX and Linux.

Figure 1 shows two screen shots of a Topos client. It depicts a workspace created by landscape architects and contains related materials such as text documents, pictures from work sites, sketches and CAD drawings. Double clicking any of the documents will launch the document in its appropriate application, and changes to the document will be reflected within Topos in near-real time. The hemisphere in the top of figure 1.a represent the top workspace, and the other spheres are sub-workspaces. These objects are not considered by the parser. The 3D objects can be moved, sized, rotated etc; light effects may be applied; documents may be made semi-transparent; manipulated as groups and so forth. The wire-frame boxes shown merely illustrate the structures found by the parser. They are in fact the computed bounding boxes of the found structures (not to be confused with the axis aligned bounding boxes of each document which will be described shortly) and are not normally shown to the user.

Grouping materials in 3D space is familiar to us all from our daily lives: we build heaps of related documents on our desks, we place related materials together on shelves, etc. A main point of the Topos prototype is to mimic and support this grouping of materials in digital form. Topos also supports grouping and organization of materials by means that are not possible in the physical world: groups can be moved and placed as a whole, sub-groups can be closed to unclutter the space etc.

Topos supports informal grouping of related documents and models within 3D workspaces. In fact, this is the basic mechanism for information management within Topos. Being three dimensional, the workspaces allow for much more freedom in placement of work materials and allows the users to oversee much larger collections of materials at a time than would be possible in two dimensions. The 3D workspaces also allow users to push groups of materials into the background to be peripherally aware of changes to them while working on something else.

The way Topos is often used, both by its developers and users (architects and landscape architects), is to create loose groups of work materials by placing materials together, and when significant groups emerge, the groups are formalized by creating sub-workspaces for them. The informal and imprecise ”direct-manipulation” way of interacting with the materials within Topos does not lead users to create nicely aligned and finely structured groups, as done in VIKI and other 2D environments. Rather, materials are typically arranged into rough structures for later fine-grained organization. One of the uses for Topos and the spatial parser by landscape architects is to sort and organize sets of pictures taken at a site visit. Within Topos a large number of pictures can be overseen at once, and the pictures can be grouped easily via the powerful direct manipulation interface. Using the parser whole groups of pictures can be manipulated at once instead of individually.

A spatial parser attempts to recognize structures and groupings from users' placements of documents, notes, images or models close to each other in space. A good parser should find the same groupings that the user thought of when creating or inspecting the groups. However, different people find different groupings in the same geometric constellations of objects. This lead us to consider parsers customized for a particular user, quite in line with (11) that provided user-customization by letting users correct the parser when it found the “wrong” structure.

Groupings in 3D space may look very different from different angles, and different in different contexts: while looked on from afar, a set of objects may seem to belong to a homogeneous group, but when looked on from a closer vantage point, and in isolation, the group may appear divided into smaller sub-groups.

Figure 1.a shows an overview of a workspace with the structures found by the parser illustrated as wire-frame bounding boxes. Figure 1.b shows a close up view of the objects appearing as a single homogeneous cluster in the bottom right part of figure 1.a. From this viewpoint the objects appear as four disjoint clusters and are recognized as such by the parser.

The above observations lead us to consider parsers that would use the user's current viewing frustum (see Figure 2) while parsing, in effect parsing the scene from the user's viewpoint. One implication of this decision is that a complete parse cannot be re-used as the user moves his viewpoint around in the virtual world. We can, however, re-use some data structures.

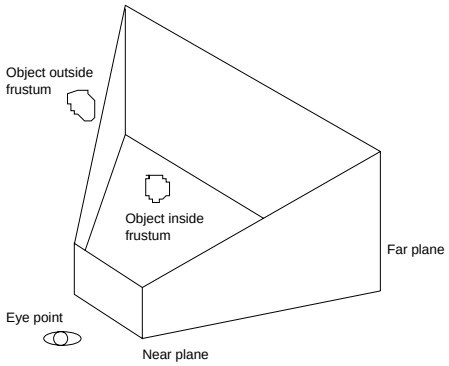


Figure 2: Viewing frustum.

One, crude, interface to the parser lets the user shift-click on an object which will engage the parser and multi-select the group of objects found around the clicked object. Shift-clicking several times on a selected group expands the selection to higher level groupings as done in VIKI. To cater for the cases where the parser does not automatically select all of the desired group, the user can select additional objects (perhaps using the parser) and add them to the current (multi-)selection. Another interface (used at an interactive whiteboard) puts a halo of icons around the most recently selected object. One of the buttons invokes the parser and allows the user to expand the selection as well.

4 An Implementation of a 3D Spatial Parser

4.1 Introduction

The basic design of our algorithm is based on the work in (11) which presented good results for 2D spatial parsing of ambiguous layouts. By ambiguous we mean that the implicit structures perceived will often vary from person to person. Figure 13 shows an example of an ambiguous layout which may be interpreted differently. The main idea of (11) is to simulate the human visual system which perceives structure in a bottom up fashion. Structures gradually emerge in our minds as we observe local object relations such as *proximity* (closeness), *similarity* (objects looking alike) and *regularity* (eg. having a regular list structure) (17).

If an algorithm is to simulate this it will have to consider not only local object relations, but also the context in which these relations exist, and it should be possible to easily adjust the algorithm to conform to the perception of a specific person.

The above observations produce a visual parsing strategy called the *link model* in (11) which we have modified and extended to meet the demands of our 3D spatial parser. In the following we will call the model the *proximity model* and talk about *edges* and *vertices* (or objects) as opposed to the links and endpoints of (11) to avoid confusion with hypertext links and endpoints as in eg. the Dexter model (18). An *edge* simulates the proximity relation and connects two objects which are *close* (The term *close* will be further elaborated later on.) The objects and edges make up the *proximity graph* which is the main data structure of our algorithm.

In perspective projection the visible part of the 3D world forms a frustum with the eye at the top of the cut-off pyramid looking towards the base (see Figure 2.) In Topos, the 3D display objects are organized into a scene tree. In order to speed up the display process of the scene tree, we do not want off-screen objects to be drawn. The process of determining what objects are visible is called frustum culling (19).

The proximity graph persists between parses and is updated whenever an object changes its characteristics such as position, orientation or size, or whenever the camera is moved, as we parse the contents of the viewing frustum seen from the user's viewpoint. Culling the scene to the active viewing frustum speeds up parsing, as fewer objects have to be considered, and gives fewer surprises to the users, as objects not visible from the current viewpoint are not considered and thus not selected by the parser.

Our algorithm is designed as a generic framework where *structure experts*, a component that specializes in finding a specific kind of structure eg. lists, can easily be plugged in to extend the parser as was first done in (3).

Briefly, a pass of the multi-pass algorithm works as follows: first edges considered too long (based on both local and global statistics of the scene) are discarded, then individual structure experts are consulted to compute levels (or strength values) for the remaining edges based on local properties. Weak edges (based on the levels) are then discarded, and pairs of conflicting edges are computed by the structure experts. Then the repress-and-reinforce phase is invoked to recompute levels for each edge based on the local level and the context of the edge. Next edges deemed too weak are again discarded. Finally, a global edge selection algorithm is invoked to select the set of non-conflicting edges which maximizes the total sum of the levels of the edges selected. The connected components in the resulting graph containing only the selected edges correspond to the structures found. The above pass is iterated as long as new structures are found and the entire parse thus produces a hierarchy of structures. In pseudo code the main loop of our implementation looks like this:

```
while(<new structure is found>) {
   discardLongEdges(proximityGraph);
   for(<each structure-expert>) {
      computeLevels(proximityGraph);
   }
   discardWeakEdges(proximityGraph);
   for(<each structure-expert>) {
      computeConflictingEdges(proximityGraph);
   }
   repressAndReinforce(proximityGraph);
   discardWeakEdges(proximityGraph);
   selectEdges(proximityGraph);
}
```

The construction and maintenance of the proximity graph as well as the above phases of the algorithm are described in detail in the subsequent sections.

Each phase of the algorithm exports a number of parameters that can be adjusted to help facilitate a specific interpretation or perception and thereby tailor the parser to a specific user's expectations.

4.2 The Proximity Graph

The proximity graph is an undirected graph where a vertex represents an object in the 3D environment and an edge represents an object relation. Two edges are said to be *adjacent* if they share a vertex.

The purpose of the proximity graph is to connect all objects that could possibly be neighbors in a structure. At the same time it is an advantage to limit the number of neighbors of each vertex as this will reduce the computations necessary in the subsequent phases of the algorithm.

The objects from the 3D environment are presented to the parser in their Local Space coordinates along with transformation matrices which make it possible for the parser to simulate the graphics pipeline. The transformations and the associated spaces are depicted in Figure 4. The View Space coordinate system can be seen in Figure 3 with the viewpoint at the origin, looking along the positive Y-axis. See the book by Foley, Van Dam, Feiner and Hughes (19) for an introduction to the graphics pipeline and computer graphics in general.

Given the bounding box of a 3D object in its Local Space, we transform this to View Space. The bounding box may be oriented arbitrarily in View Space, so next we find the axis aligned bounding box of the transformed local bounding box. For each such axis aligned bounding box we partition all other objects in View Space into the surrounding 26 $(9 + 8 + 9)$ disjoint subspaces that result from partitioning the space by the six (infinite) planes defined by the faces of the bounding box. This is a direct extension to 3D of the partitions shown in Figure 5 and used in (11).

The idea is to connect to the closest object (or objects, if several objects are equally distanced) in each partition. In this way insignificant objects far away will not be connected to the space partitioning object. A neighboring object may span several partitions, so we have to determine in which partition such an object is to be connected. As distance-measure we use the shortest 3D Euclidian distance between the two transformed bounding boxes in View Space. This distance specifies a point on each bounding box (in degenerate cases (parallel objects) a single point is not determined) and we use the point on the neighboring

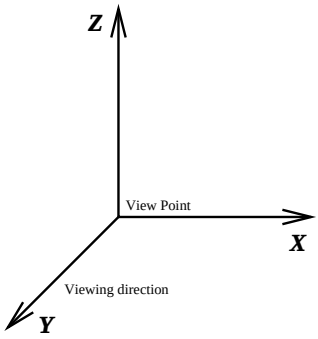


Figure 3: The viewing coordinate system.

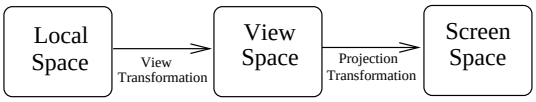


Figure 4: The transformations and associated coordinate spaces.

object to determine in which partition it is connected. We have considered other options such as using the center of mass as a means of partitioning, but as shown in Figure 5 this may result in an object being connected in a partition where the objects are not necessarily close.

As can be seen from Figure 6 the projection of an edge representing the shortest distance in View Space is not necessarily the shortest distance in Screen Space. When working in Screen Space we must use the shortest distance in Screen Space (and not just the length of the projection of the edge representing the shortest distance in View Space) as this corresponds to what the user actually perceives as being the shortest distance. We find the shortest distance between two objects in Screen Space by projecting the vertices of their bounding boxes onto the screen, next computing their respective convex hulls in 2D and then finding the shortest 2D distance between these convex hulls.

4.3 Computing Levels

A structure expert for structure type τ (eg. **list**) assigns each edge e in the proximity graph a *level* λ_e^τ of type τ. The level λ_e^τ expresses to what extent the two vertices of the edge can be seen as part of a structure

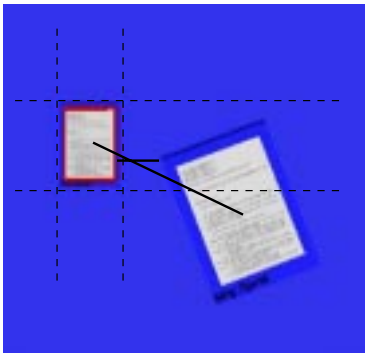
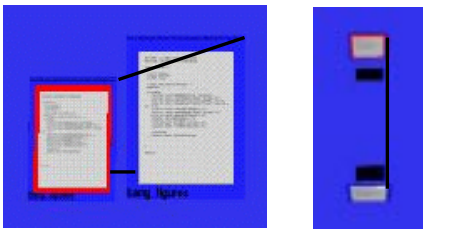


Figure 5: The difference between using the points defining the shortest distance and the center of mass as a means of partitioning.



a)Front view b) Top view

Figure 6: The effect of the projection transformation. Objects with the same (X, Z) coordinates, but different Y coordinates may appear next to each other on the screen.

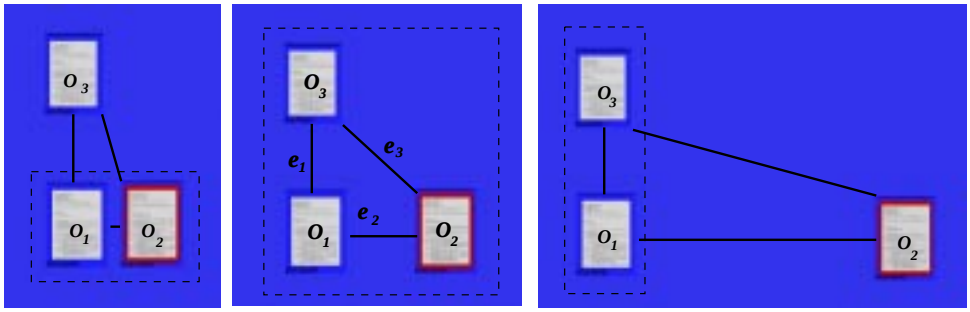


Figure 7: Horizontal and vertical lists versus clusters.

of type τ. It can be used to encode characteristics such as regularity and similarity, and it is intended merely as a local measure, no surrounding edges or objects should be taken into consideration when computing it. It is important for the subsequent phases of the algorithm that the levels are comparable independently of which structure expert computed them, and the levels are therefore restricted to lie in the interval $[0; 1]$. If λ_e^τ is greater than zero the edge is said to have type τ. Contrary to (11) an edge can have multiple types as it is not possible to restrict an edge to a single type based on the level alone, the context has to be taken into consideration before this judgment can be made. In Figure 7.a we want O_1 and O_2 to constitute a list, and in Figure 7.b we want O_1, O_2 and O_3 to be a cluster. This would not be possible if e_2 was assigned a single type, eg. **list**, based on the largest level associated to it.

Our algorithm can simulate allowing only one type by letting the *Discard Weak Edges* phase keep only the maximum level and its associated type on each edge.

A structure expert for type τ has a number of parameters which influence the level computed. A parameter could eg. be the distance between two objects. Each parameter, p_i, has an associated significance, $s_i \in [0; 1]$, which describes how significant the parameter is in the computation of the level. Given an edge e, a structure expert first computes the values, v_i, of all parameters, p_i. Typically, v_i will lie in the interval $[0; 1]$. Then the level is computed as

$$\lambda_e^\tau = clamp(\prod_{i=1}^{n} signify(s_i, v_i)) \tag{1}$$

where *signify* is defined as $signify(s_i, v_i) = s_i v_i + 1 - s_i$, and *clamp* clamps a value to the unit interval (some times called saturation: values below zero are clamped to zero, values above 1 are clamped to 1). Given a parameter, p_i, if $s_i = 0$, *signify* is 1 and has no significance in (1), if $s_i = 1$, *signify* is v_i.

The significance of all the parameters can be set by the user to adjust the algorithm to suit his or her own interpretation. As opposed to the implementation in (3) the order in which the structure experts are applied is not significant. In the following sections we describe the parameters of each of our current structure

Figure 8: The coordinate system of an edge.

experts.

4.3.1 Cluster Expert

A cluster is merely a rough grouping of objects and as a consequence we want the level to express to what extent objects are located near each other. For each edge, e, the cluster expert uses the following functions to compute parameter values:

- $1 - length2D(e)$, where *length2D* is the shortest distance in Screen Space.
- $1 - abs(normalize(y_1) - normalize(y_2))$ where y_i is the Y coordinate of vertex i of edge e, and *normalize* maps a coordinate to $[0; 1]$ based on the dimensions of the frustum. A low significance of this parameter will facilitate groupings of objects far from each other in Y coordinates (the viewing direction) but which may appear close on the screen. A high significance can be used to ignore objects far away.

Instead of using the two parameters above one could use a function of the length of the edge in View Space. By using that strategy it would be harder to control the groupings of objects which seem to be close in Screen Space because of the viewing direction.

Parameters based on object type, color and document similarity may of course also be considered, but for this paper we have chosen to focus on the geometric parameters.

4.3.2 List Expert

Contrary to (11) and (3) a list in a pure 3D environment can be oriented arbitrarily, as objects can be rotated and scaled freely. As a consequence of the above the list expert works in the coordinate system **(a, b, c)** defined by the three vectors:

a The vector representing the shortest distance in Screen Space, see Figure 6.

b The viewing direction

c The cross product, $a \times b$.

Figure 8 shows an edge, its vertices and the corresponding coordinate system. The vector, **a**, is said to define the direction of the list.

For each edge, e, the list expert computes the same parameter values as computed by the cluster expert in addition to the following parameter values:

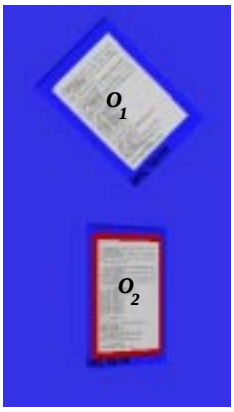


Figure 9: Lists and object rotation.

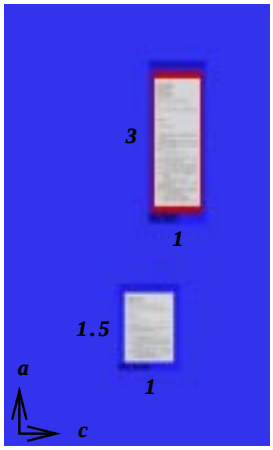


Figure 10: The length of the objects are indicated on the sides.

- The rotation of each of the vertices of the edge about each of the **(a, b, c)** axes. Let θ_{ij} be the rotation of vertex i of edge e about axis j, where j is one of **a**, **b** or **c**. If $\theta_{ij}\ mod\ 90° < 45°$, we compute it as $1 - (\theta_{ij}\ mod\ 90°)/45°$, otherwise as $1 - (90° - \theta_{ij}\ mod\ 90°)/45°$. In Figure 9, O_1 is rotated $45°$ about the **b** axis and O_2 is rotated $20°$ about the **a** axis.
- The ratio of the extents of the objects in Screen Space projected orthogonally on the **a**-axis, computed as $\frac{extent_{min}}{extent_{max}}$. Figure 10 shows a ratio of $\frac{1}{2}$.
- As above for the **c**-axis. Figure 10 shows a ratio of 1.
- The ratio of the overlap of the two objects projected orthogonally on the **c**-axis. Figure 10 shows an overlap ratio of $\frac{1}{2}$.
- The rotation of the direction of the list about the viewing direction. Let ϕ_l be the the rotation of edge e about the viewing direction **b**. If $\phi_l\ mod\ 90° < 45°$, we compute it as $1 - (\phi_l\ mod\ 90°)/45°$, otherwise as $1 - (90° - \phi_l\ mod\ 90°)/45°$. In Figure 8 the direction of the list is rotated $45°$ about the viewing direction. This parameter is included as people have a tendency to look for lists that are either vertical or horizontal.

If a user finds any of the above parameters to be insignificant he or she can just set the corresponding significance value to zero.

4.4 Discarding Weak And Long Edges

To remove edges that are considered too long, we first remove edges that are longer than a user specified *maximum edge length*. Next we perform the following two tests in each connected component in the resulting proximity graph (as edges longer than the *maximum edge length* are removed several connected components may arise): if the length of the edge is longer than a user specified rate times the standard deviation plus the mean value of all edge lengths, the edge is removed (this computation is done in both View and Screen Space); if the length of an edge is longer than a user specified rate times the minimum length of an adjacent edge plus a user specified term, the edge is also removed (this computation is done in Screen Space). The purpose of removing long edges is to avoid edges between objects that are far from each other in partitions where there are no "close" objects and thus split the proximity graph into more connected components which will speed up the *Edge Selection* phase.

A level λ_e^τ is considered *weak* if the ratio of λ_e^τ to the maximum level of all types of all edges adjacent to e is below a user specified threshold (thresholds and other parameters can be set using a dialog box), or if λ_e^τ itself is below a user specified threshold. Weak levels are removed from the edge. If all levels are removed from an edge, the edge itself is removed. Removing weak edges serve to avoid influences from weak edges in the *Repression And Reinforcement* phase and helps partition the proximity graph into smaller connected components.

4.5 Computing Conflicting Edges

The purpose of this phase is to compute edge conflict constraints that are used in the *Edge Selection* and *Repression And Reinforcement* phases.

An edge conflict can exist between two adjacent edges and it is associated to a type on each edge. Formally we can describe it as a tuple, (e, τ, e', τ') where e and e' are edges and τ and τ' are types of e and e' respectively.

During the ensuing *Edge Selection* phase an edge is either selected and assigned a single type (from its multiple types) or simply discarded. In Figure 7.b, if e_1 is selected and given type **list**, then e_2 cannot be selected and assigned type **list** and vice versa, as this would require O_1 to be part of both a vertical and a horizontal list. The edges e_1 and e_2 are therefore said to be in conflict wrt. type **list**. The type of edge conflicts where τ equals τ' are computed by the structure experts in turn eg. the list expert computes all conflicts wrt. type **list**.

Two edges can never be in conflict wrt. type **cluster**, but as already indicated two edges can be in conflict wrt. type **list**.

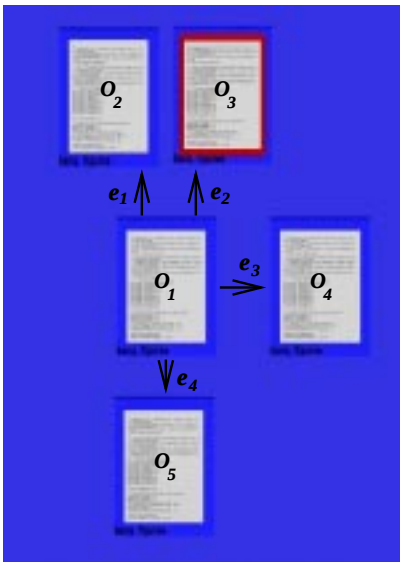


Figure 11: Examples of conflicting edges.

If an edge has type τ and an adjacent edge has a different type τ' the edges are said to be in conflict wrt. (τ, τ') as an object cannot be part of two differently typed structures at once. This type of edge conflict is called an implicit edge conflict constraint and is automatically enforced during the *Edge Selection* phase.

The list expert looks at all pairs of adjacent edges in the proximity graph which have type **list**. For each such pair a *conflict factor*, $c \in [-1; 1]$, is calculated. A value less than or equal to zero indicates that the edges are in conflict, a value of -1 expresses maximum conflict. Note that an implicit conflict constraint is interpreted as a maximum conflict ie. it has $c = -1$.

The judgment whether two edges are in conflict is based on the angle, ϕ, between the vectors representing the shortest distances in Screen Space and is parameterized by two angles, ϕ_1 and ϕ_2, with $\phi_1 < \phi_2$. If $\phi \leq \phi_1$ the conflict factor is -1, if $\phi \geq \phi_2$ the conflict factor is 1, otherwise if $\phi_1 \leq \phi \leq \phi_2$ the conflict factor is interpolated linearly between -1 and 1. In Figure 11, if $\phi_1 = 100°$ and $\phi_2 = 120°$, (e_1, e_2), (e_1, e_3) and (e_2, e_3) are in conflict, but eg. (e_1, e_4) are not in conflict.

4.6 Repression And Reinforcement

The levels computed by the structure experts are merely a local measure of how well the objects form part of a structure. The purpose of this phase of the algorithm is to recompute all the previously found levels based on the global context of each edge. As mentioned earlier, including the context of objects in the computation of structure in ambiguous 3D environments is a necessity.

The repression and reinforcement technique (11) is based on a few simple observations:

- Edges in conflict should repress each other as they correspond to different interpretations.
- Edges not in conflict should reinforce each other as the objects they connect can be interpreted as part of the same structure.

See (11) for a more thorough explanation.

For each edge, e, in the proximity graph we in turn look at its levels λ_e^τ. When considering a specific level, we see e as if it had only one type, and we now want the context of the edge to influence the level. By context we mean all edges adjacent to e. We want all interpretations to influence the level at the same time. For example, assume we are looking at type **list** of edge e ($\lambda_e^{\mathbf{list}}$) and considering edge e_i adjacent to

e. If e_i has type **list** and e and e_i are not in conflict wrt. type **list** they should reinforce each other. On the other hand, if e_i also has type **cluster** it should at the same time repress e.

For each edge, e_i, adjacent to e we compute the term, $I_{i,j}$ for each $\lambda_{e_i}^{\tau_j}$ of e_i, where j ranges over the types of edge e_i. If the conflict factor, c, (wrt. the edge conflict (e, τ, e_i, τ_j)) is greater than zero, $I_{i,j}$ is computed as

$$I_{i,j} = REINFORCE\_RATE_{\tau,\tau_j}\, c\lambda_{e_i}^{\tau_j}$$

otherwise it is computed as

$$I_{i,j} = REPRESS\_RATE_{\tau,\tau_j}\, c\lambda_{e_i}^{\tau_j}$$

The final level, $\hat{\lambda_e^\tau}$, for e is then computed as

$$\hat{\lambda_e^\tau} = \lambda_e^\tau (1 + IR \sum_{i=1}^{n} \sum_{j=1}^{m_i} I_{i,j})$$

where n is the number of edges adjacent to e and m_i is the number of types of e_i. Each of the factors $REINFORCE\_RATE_{\tau,\tau'}$, $REPRESS\_RATE_{\tau,\tau'}$, and the interference rate, IR, can be adjusted by the user.

4.7 Selecting Edges

In this phase each edge is either selected and assigned a single type (from one of its multiple types) or the edge is discarded. The edge selection strategy has to select and discard edges based on the types and levels of each edge whilst respecting the edge conflict constraints. As we have a generic framework into which structure expert components can be plugged the algorithm should be able to find structures that are arbitrarily large and complex. We have therefore chosen a global edge selection strategy that maximizes the total sum of levels corresponding to the edges selected and the types assigned. This is contrary to the local edge selection strategy chosen in (11).

For each connected component in the proximity graph our edge selection strategy selects the group of edges $(e_1, e_2, \ldots, e_n)$ and corresponding types $(\tau_1, \tau_2, \ldots, \tau_n)$ for those edges such that the sum $\sum_{i=1}^{n} \hat{\lambda_{e_i}^{\tau_i}}$ of levels assigned to the selected edges is maximized and the edge conflict constraints not violated.

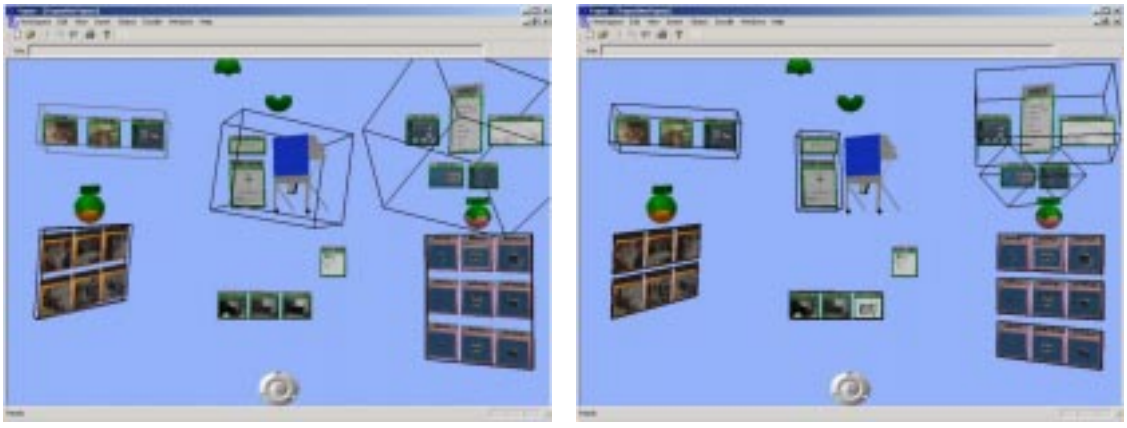
Unfortunately, this edge selection strategy is NP-hard as can be seen by applying a simple polynomial time reduction from *ONE-IN-THREE-SAT* where all literals are positive. Currently we use a *branch and bound* algorithm which computes bounds based on the levels.

We have also implemented a greedy edge selection strategy which runs in time $O(m^2)$ where m is the number of edges. It works by repeatedly selecting the edge with the largest level and assigning it the corresponding type. When an edge is selected the adjacent edges not yet selected or discarded are either constrained to have the type of the edge just selected or discarded if they were constrained in a previous iteration to have a type different from the type of the edge just selected. The greedy selection strategy seems to produce good results.

The time used by the selection algorithm may be reduced by adjusting the user definable thresholds which may cause the proximity graph to split into smaller connected components.

When all edges have been processed this way each connected component in the resulting proximity graph corresponds to a structure in the 3D environment. For each such structure we find an approximation to the smallest bounding box enclosing it (20). Next we build an additional proximity graph whose nodes are the structures found and the objects which did not form part of any structure. The parser continues to look for hierarchical structures this way until no new structures are found. This tree of structures forms the result of the spatial parse.

Figure 12: Shows how the user interacts with the parser.



(a) (b)

Figure 13: An ambiguous workspace layout parsed with two different parameter settings. In (a) the workspace is parsed using the “Parse for all structures” parameter setting, and both clusters and lists are found. In (b) the workspace is parsed using the “Parse for lists” parameter setting.

5 A Simple 2D Parser for 3D Scenes

The true 3D spatial parser is rather complicated and consists of a large amount of code. This section describes an alternative, and simpler, spatial parser for 3D scenes that we have also implemented within Topos, and compares it to the full 3D parser described above.

In Topos the cull phase works by computing approximate bounding boxes for all sub-trees in the scene graph tree. Sub-trees with bounding boxes completely outside the viewing frustum are marked and will not be sent further down the graphics pipeline. During the cull phase we also compute 2D axis aligned bounding boxes for the on-screen objects. These are used by the 2D spatial parser. Computing the 2D boxes does not add significantly to the cull phase.

The main step of the 2D spatial parser consists of drawing the axis-aligned 2D bounding boxes from the cull phase into a low resolution 100×100 pixel grid. Each pixel in the low resolution grid consists of a list of object IDs. Rasterizing a box onto a pixel appends the object ID of the corresponding 3D object to the list. Drawing only the axis-aligned boxes significantly simplifies and speeds up the rasterization. An alternative implementation might use a list of rectangles together with an algorithm for finding intersections among the rectangles instead of rasterizing the rectangles. Our particular implementation was primarily chosen because it was easy to implement. As most objects in Topos workspaces are thin 3D boxes representing 2D documents, their projected 2D bounding box does not usually differ significantly (for our purpose) from their true 3D rendition.

Before rasterizing the 2D boxes onto the grid, the boxes are scaled around their center according to a scale factor. Scaling up the 2D boxes makes objects, which do not overlap on screen, overlap in the low resolution grid. Increasing the scale factor makes the 2D boxes larger and thereby makes larger and larger regions overlap. We increase the scale to find larger and larger clusters to support hierarchical click selection.

To find clusters in the grid we employ a simple breadth-first flood-fill algorithm, that given a start-pixel (computed from a mouse position) floods the region of connected boxes in the low-resolution grid, while collecting the object IDs of corresponding 3D objects. The algorithm works very well to select larger and larger clusters of nearby objects.

Our approach to scaling the 2D boxes increases the box size relative to the original dimensions of the box, so if the original box has dimensions (w, h), the n times scaled box has dimensions $((1 + n\delta)w, (1 + n\delta)h)$. This makes tall, slim boxes grow faster vertically than horizontally, and vice versa for low, wide boxes. An alternative would be to grow the 2D boxes by a constant number of 2D pixels in each dimension, such that the n times scaled box has dimensions $(w + n\delta', h + n\delta')$. Yet another variation of the two scaling methods above would be to grow the scale factor super linearly. This would enable faster, but less accurate, selection of larger clusters.

Our 2D spatial parser of 3D scenes can also be extended to take depth information into account by using a Z (or depth) buffer as in conventional 3D graphics. By associating a depth value with each pixel in the low resolution grid one can avoid far objects from being included in a cluster based on the depth. The depths are being computed during the cull phase already, so this extension would not add much to the parsing time.

The simple 2D approach described here is strictly view-dependent and does not lend itself easily to collaborative spatial parsing as explored by CAOS. Also, the 2D spatial parser is so simple and efficient that making it collaborative to share the computations would be meaningless. Using a constant grid size the complexity of the 2D parser is linear in the number of objects in the viewing frustum, which is the best one can hope for.

Comparing the simple 2D parser to our true 3D parser highlights a number of key differences: The 2D parser is simple taking around 300 lines of C++ code by virtue of being tightly integrated with the rest of the system, taking advantage of the existing culling code. The 3D parser is more complex, taking up around 10000 lines of C++ code. It is, however, more stand-alone and thereby easier to separate from the particular Topos implementation. The 2D parser is able to find only *clusters* of nearby objects, whereas the 3D parser finds arbitrarily oriented lists as well as clusters (and can be extended to find additional structures by plugging in new structure experts), and is able to prioritize among the different potential structures found.

The 3D parser is highly configurable, and can be configured to very closely match the 2D parser (finding

the same clusters in the same scenes). This is achieved simply by turning down the sensitivity to depth information and turning off the list experts. In this sense the 3D parser can be viewed as a generalization of the 2D parser, even though their implementations differ significantly.

The simplicity and usefulness of the 2D parser indicates that spatial parsing could and should be integrated into many 3D environments (3D modelers, VR environments etc.).

6 Discussion

In pure 2D, there is no real concept of a view point. However, a 2D interface with scrollbars, where the user only sees part of the canvas at a time, might benefit from a spatial parser taking the visible area into account. In a zoom-able 2D interface or a 2D interface with multiple fish-eye views a spatial parser might also benefit from taking the current viewpoint, as well as the placements of the fish-eye views into account. However, changing the 2D view point does not affect the apparent clustering of objects as much as changing the view point in a true 3D world.

The decision whether the spatial parser should parse the apparent – subjective – view or some objective reality of a complete workspace depends on the intended application of the parser results. In 2D the difference between objective and subjective placements of objects is fairly small, but in 3D the difference can be large, as some of the figures have shown. We think that parsing our 3D workspaces from an objective global view point would much too often confuse users as structures which seem apparent on the screen do not necessarily exist in View Space, and would result in them struggling to place materials precisely to fit the parser instead of getting on with their inevitably subjective work.

Topos is currently geared for open, unrestricted spaces that are sparsely furnished. The reasons for this are to some degree accidental, but are also a result of trying not to limit the users arbitrarily: it is up to the users themselves to furnish their spaces. Also, providing pre-fabricated background geometry would serve to limit the spaces and thus work against the goal of providing more screen real estate via the use of 3D. Naturally, our parser is therefore also geared toward open, sparsely furnished spaces. In more closed surroundings a parser taking the background geometry into account would be more natural.

When *typical* workspaces become furnished with background models and “landscaping”, we suppose that users will gradually build up a ”feel for the space”, getting to know more about the positions of materials in relation to the surrounding “landscape”. In such a scenario a parser taking world coordinates and background geometry into account would be more useful as it would correspond better to the model of the space inside the users head. However, a drawback of such an approach would be that users unfamiliar with a given workspace would feel less at ease with the parser until they had understood – “got the feel of” – that particular workspace. In effect we suppose that as the user gets more familiar with a space, he is liberated from the restrictions of his particular viewpoint by having a model of the space present in his mind at all times. This, however, presupposes that the user moves around while learning the space so as to build a more global model of the space inside his head. Our limited observations of current use practice tend to show great variations in this respect: some users move around a lot, others stand still and instead manipulate the materials, eg. to view them up close. Use practice also differs from workspace to workspace: in workspaces shared among several users social conventions stipulate that materials are to be left in their place so that others can find them again, and this forces the user to move around to look closely at materials. In personal workspaces users tend to move materials around more than themselves, and thereby, maybe surprisingly, build less global models in their heads about spatial relationships than they do in shared workspaces.

Having large 3D models serve as “landscaping” and background also presents a technical challenge as the 3D models would typically be non-convex and therefore harder to handle by the parser which currently approximates models by a (convex) bounding box. A first approach to a solution to this would be to let the parser treat the models as a set of convex polytopes instead of one large model, however, this will be more computationally intensive than the current approach. On the other hand, the current parser would be easy to adapt to a parsing strategy based on global coordinates, simply by plugging in new structure experts.

Figures 12 and 13 show a real workspace used within the project group. Figure 12 shows how users interact with the parser in the current implementation. When an object is selected various tools appear as icons surrounding it. Clicking the rippled circle icon just above the object pops up a menu from which the

parser can be activated with different parameter settings. Currently we support three predefined parameter settings, “Parse for all structures”, “Parse for lists” and “Parse for clusters”. Having these predefined settings enables the user to directly use the flexibility of the parser without having to adjust parameters every time he desires to parse the ambiguous workspace layout in a different way. Using the “Parse for all structures” parameter setting the parser tries to find both lists and clusters in the workspace layout. Using the “Parse for lists” parameter setting the parser favors lists over clusters and vice versa for the “Parse for clusters” parameter setting. Figure 13 shows the same workspace parsed with the two predefined parameter settings “Parse for all structures” and “Parse for lists” respectively. The bounding boxes are merely shown to indicate which structures are found by the parser. Normally only the objects in the same structure as the object from which the parser was invoked are selected and highlighted. Only the structures in one hierarchical level are shown. Clicking the rippled circle icon again expands the selection to higher level groupings. When giving a presentation and pulling forward groups of objects or manipulating groupings in general it is often useful to be able to parse the workspace layout in several ways. In the bottom right part of the workspace in Figure 13 nine screen shots describing various interaction modes supported by Topos are situated. All objects are related but the objects in each row are more significant to each other as they describe different aspects of the same interaction mode. As a result the objects are placed such that their constellation can be interpreted both as a cluster and as three distinct lists. In Figure 13.a the parser recognizes the nine objects as a single cluster, in Figure 13.b they are recognized as three lists. This enables users to select all nine objects or just one row.

Figure 13 only illustrates part of the flexibility of the parser. By adjusting parameters the parser can also be configured to split apparent structures in screen space into more structures if the objects are far from each other in the viewing direction etc. Also note from figure 13.a that the parser takes not only documents but also 3D models into consideration.

An alternative way of selecting multiple objects – not used in Topos – is to use sweep-selection: sweeping out a box around a group of objects to select them. This, however, works best in a pure 2D interface as the groups formed in 3D seldom coincide with an axis aligned 2D box. Also a 2D box could either select objects at a certain depth range or all the objects visible through the box.

No formal user tests have yet been carried out, but the users of the Topos system that have tried to use the parser have given very positive feedback. The parser allows treating informal groups of objects as a whole without making their relationship formal and more permanent by explicitly grouping the objects into a sub-workspace. We also see a tendency of users adapting to the parser: rearranging structures slightly to get the “right” parse. To some degree this may be beneficial in that this helps users to structure their workspaces similarly and thereby making the use of a workspace easier for others.

Currently, the 3D parser performs acceptably for interactive use, having sub-second response times for typical scenes (10s of documents) on a 500 MHz PC. The parser maintains its data structures while objects are added, moved, deleted and while changing the viewpoint. During interactive movements of objects the parser is not updated until the mouse button is released.

Our 3D spatial parser is a generic class library, thus independent of Topos, and communication is done via abstract interfaces. This means that the parser can easily be incorporated into any 3D environment that can be made to support and use the interface of our parser. The parser has access to objects in the 3D environment by means of proxy objects through which the viewing matrix, bounding box, object type etc. can be obtained. The required interface of a proxy object is listed below:

```
class Proxy {
  public:
    virtual void getViewMatrix(MATRIX *m);
    virtual void getBBox(BBOX *bbox);
    virtual std::string getType();
};
```

A subset of the abstract interface provided by the parser is:

```
class Parser {
  public:
    virtual void init();
    virtual void exit();
    virtual Tree *parse();
    virtual GRAPH *getProximityGraph();
    virtual void setProjMatrix(MATRIX *m);
```

```
    virtual void setNearFar(double n, double f);
    virtual void addProxyObject(Proxy *proxy);
    virtual void remProxyObject(Proxy *proxy);
    virtual void removeAll();
    virtual void update(Proxy *proxy);
    virtual void updateAll();
};
```

Topos uses the parser as a class library linked into the executable. One might also experiment with implementing the parser in a shared middle-ware layer as suggested in (10). This would entail implementing the above interface eg. as CORBA interfaces. However, the advantages of such a solution are questionable, as quite a lot of calls to the proxy interface would have to cross the network.

In the shared 2D parser of CAOS (9) the point was to make the parser incremental and share the parse result among many cooperating users. We do not think that such sharing would be beneficial for our 3D parser as cooperating users (fully supported by Topos) do not typically share the same view point, and would therefore require different parses. Also, users cooperating with other Topos users may only be sharing parts of the scene and as a whole the scene seen by one user may be very different from the scene seen by another.

As mentioned before our 3D parsing strategy is strongly influenced by the algorithm described in (11). Another approach would be to take departure in data clustering as known from data mining applications, as for example (21). However, most data clustering algorithms use rather simple metrics as the basis for clustering, and do not cope with conflicting relations as do the algorithms described here.

7 Conclusion and Future Work

We have described the implementation of a 3D spatial parser component to the Topos system which can be extended to recognize additional structures by plugging in new structure experts. It has proven flexible and adjustable to conform to various specific user's interpretation in ambiguous 3D layouts. We also described a 2D spatial parser for 3D scenes which is easy to implement, and compared the two parsers.

A number of developers and users have used the spatial parser and provided positive feedback, but no formal user tests have been carried out yet.

In the future we plan to develop and test additional structure experts to verify that our framework is capable of handling arbitrary structures.

A structure expert cannot ensure that a certain edge is selected as edges are selected independently in the *Edge Selection* phase. As a result a structure must have the property that it can be split into two structures of the same type at an arbitrary edge (which is true for lists and clusters). This property can be eliminated by enforcing additional constraints in the *Edge Selection* phase, and we are currently implementing this.

As regards efficiency we could obtain a good deal by making not only the proximity graph maintenance, but also the parsing process incremental. This could be done by including all phases except *Edge Selection* of the parsing algorithm in the incremental proximity graph maintenance and at the same time keep a version of the most recent parse hierarchy. We could then parse, or equivalently select edges, only in the connected components in the graph that were affected by the changes or optimally only in the union of objects with which the altered objects interfere as is done in (9). For complex structures though, the latter approach is not likely to work in the proximity model, but it could be used to produce faster approximations in very complex workspaces. Another optimization would be to improve the space partitioning strategy described in section 4.2.

An obvious extension would be to build consideration of visual cues such as object type and color etc. into the parser.

The parameters of the parser which can be set using a dialog box can be hard to tune by hand and as a consequence we are already walking in the footsteps of (11) and trying to add a notion of evolutionary algorithms to allow the parser to learn from feedback provided by users, and thus implicitly adjust the parser parameters. An other route would be to provide a more intuitive dialog to let the user adjust the many parameters. One could imagine a dialog with a small preview of some objects with found structures shown. As the user adjusted the parameters the parser would re-parse the test scene and show the results immediately.

Acknowledgments. We want to thank professor Kaj Grønbæk and the rest of the WorkSPACE group for many fruitful discussions and good advice about this paper. We would also like to thank Anders Brodersen for helping integrating the parser into Topos. This work has been funded by the Danish Research Council's Centre for Multimedia (project no. 9600869), the EU Esprit LTR project DESARTE and the EU IST project WorkSPACE.

References

[1] CATHERINE C. MARSHALL, FRANK G. HALASZ, RUSSEL A. ROGERS, and WILLIAM C. HANSSEN JR. Aquanet: A hypertext tool to hold your knowledge in place. In *Proc. ACM Hypertext'91*, pages 261–275. ACM, ACM Press, December 1991.

[2] CATHERINE C. MARSHALL and RUSSEL A. ROGERS. Two years before the mist: Experiences with aquanet. In *Proc. ACM ECHT'92*, pages 53–62, Milano, November 1992. ACM, ACM Press.

[3] CATHERINE C. MARSHALL, FRANK M. SHIPMAN, and JAMES H. COOMBS. VIKI: Spatial hypertext supporting emergent structure. In *Proc. ACM ECHT'94*, pages 13–23. ACM Press, September 1994.

[4] MONIKA BÜSCHER, MICHAEL CHRISTENSEN, KAJ GRØNBÆK, PETER KROGH, PREBEN MOGENSEN, DAN SHAPIRO, and PETER ØRBÆK. Collaborative augmented reality environments: Integrating VR, working materials and distributed work spaces. In *Proc. Collaborative Virtual Environments 2000*, pages 47–56, San Francisco, September 2000. ACM, ACM Press.

[5] MONIKA BÜSCHER, MICHAEL CHRISTENSEN, PREBEN MOGENSEN, DAN SHAPIRO, and PETER ØRBÆK. Creativity, complexity and precision: Information visualization for landscape architecture. In *Proc. ACM Intl. Symp. on Information Visualization 2000*, pages 167–171, Salt Lake City, October 2000. ACM, ACM Press.

[6] PREBEN MOGENSEN and KAJ GRØNBÆK. Hypermedia in the virtual project room – toward open 3D spatial hypermedia. In *Proc. 11th Conf. on Hypertext and Hypermedia*, pages 113–122, San Antonio, TX, 2000. ACM.

[7] BENJAMIN-B. BEDERSON and JAMES D. HOLLAN. Pad++: A zooming graphical interface for exploring alternate interface physics. In *Proc. ACM UIST'94*, pages 17–26. ACM, ACM Press, 1994.

[8] CATHERINE C. MARSHALL and FRANK M. SHIPMAN. Searching for the missing link: Discovering implicit structure in spatial hypertext. In *Proc. ACM Hypertext'93*. ACM, ACM Press, November 1993.

[9] OLAV REINERT, DIRK BUCKA-LASSEN, CLAUS A. PEDERSEN, and PETER J. NÜRNBERG. CAOS: A collaborative and open spatial structure service component with incremental spatial parsing. In *Proc. ACM Hypertext'99*, pages 49–50, Darmstadt, 1999. ACM, ACM Press.

[10] PETER J. NÜRNBERG, JOHN J. LEGGETT, and ERICH R. SCHNEIDER. As we should have thought. In *Proc. 8th. ACM Conf. on Hypertext*, pages 96–101, Southampton, UK, April 1997. ACM, ACM Press.

[11] TAKEO IGARASHI, SATOSHI MATSUOKA, and TOSHIYUKI MASUI. Adaptive recognition of implicit structures in human-organized layouts. In *Proc. 11'th Intl. Symp. on Visual Languages (VL'95)*. ACM, ACM Press, 1995.

[12] JOCK D. MACKINLAY, G. G. ROBERTSON, and S. K. CARD. The perspective wall: Detail and context smoothly integrated. In *Proc. CHI'91*, pages 173–179, New Orleans, 1991. ACM.

[13] FRANK M. SHIPMAN, CATHERINE C. MARSHALL, and MARK LEMERE. Beyond location: Hypertext workspaces and non-linear views. In *Proc. ACM Hypertext'99*, pages 121–130, Darmstadt, February 1999. ACM, ACM Press.

[14] S. K. CARD, G. G. ROBERTSON, and W. YORK. The WebBook and the Web Forager: An information workspace for the world-wide web. In *Proc. ACM SIGCHI'96*, pages 111–117. ACM, ACM Press, April 1996.

[15] GEORGE G. ROBERTSON, MARY CZERWINSKI, KEVIN LARSON, DANIEL C. ROBBINS, DAVID THIEL, and MAARTEN VAN DANTZICH. Data mountain: Using spatial memory for document management. In *ACM Symposium on User Interface Software and Technology*, pages 153–162, 1998.

[16] G. ROBERTSON, M. V. DANTZICH, D. ROBBINS, M. CZERWINSKI, K. HINCKLEY, K. RISDEN, D. THIEL, and V. GOROKHOVSKY. The task gallery: A 3D window manager. In *Proc. ACM SIGCHI'00*, pages 494–501. ACM, ACM Press, 2000.

[17] ILONA ROTH and JOHN. P. FRISBY. *Perception and Representation: A cognitive approach (Open Guides to Psychology)*. The Open University, 1986.

[18] FRANK HALASZ and MAYER SCHWARTZ. The Dexter hypertext reference model. *Communications of the ACM*, 37(2):30–39, February 1994.

[19] J. D. FOLEY, A. VAN DAM, S. K. FEINER, and J. F. HUGHES. *Computer Graphics: Principles and Practice*. Addison-Wesley, Reading, MA, 1990. 2nd edition.

[20] D. MANOCHA S. GOTTSCHALK, M. C. LIN. OBBTree: A hierarchical structure for rapid interference detection. In *Proc. ACM SIGGRAPH*, pages 171–180, New Orleans, USA, August 1996. ACM, ACM Press.

[21] SUDIPTO GUHA, RAJEEV RASTOGI, and KYUSEOK SHIM. ROCK: A robust clustering algorithm for categorical attributes. In *Proc. 15'th Intl. Conf. on Data Engineering 1999*. ACM, ACM Press, 1999.