

A system for Virtual Directories using Euler Diagrams

Rosario De Chiara¹

*ISISlab - Università degli Studi di Salerno
84081, Baronissi (Salerno), Italy*

Mikael Hammar²

*Apptus Technologies AB
IDEON research centre
SE-223 70 Lund, Sweden*

Vittorio Scarano¹

*ISISlab - Università degli Studi di Salerno
84081, Baronissi (Salerno), Italy*

Abstract

In this paper, we describe how to use Euler Diagrams to represent *virtual directories*. i.e. collection of files that are computed on demand and satisfy a number of constraints. We, then, briefly describe the state of VENNFS project that is currently modified to include this new capability. In particular, we show a data structure designed to answer queries about a given Euler Diagram and its sets. The data structure *EulerTree* described here is based on the R-Tree (see [1]), a data structure designed for answering *range queries* over a family of shapes in the 2-dimensional space.

Key words: Euler Diagrams, File Systems, HFS,R-Tree,
EulerTree

1 Introduction

File access and, in general, file management is the most common task in daily use of personal computers. The way in which file accessing and categorization is performed is strongly influenced by how the file system itself is designed.

¹ Email: {dechiara, vitsca}@dia.unisa.it

² Email: Mikael.Hammar@apptus.com

The pattern followed in designing file systems, even modern ones, is the “hierarchical file system”, HFS for short, in which files are categorized in folders, and folders can be put inside each other, creating a tree shaped structure.

The idea of categorizing information inside a hierarchy is intuitive and easy to understand even by people not fond of computers. Since the creation of the early versions of HFS, the theoretical design has been garnished with a daily life metaphor that greatly contributed to the diffusion of it: the *office metaphor*.

The metaphor simply uses the concept of a *filing cabinet* to symbolize the mass storage, high-level directories are represented by the drawers, lower-level subdirectories may be represented as file folders within the drawers.

The once inspiring metaphor of the office quickly got old and a limitation [2]: “*The way to advance the interface is not to develop ever-more-faithful imitations of the desktop, but instead to escape the limitations of the desktop.*”. A paradigmatic example of this kind of mimicking is the definition of what a directory is (see [3]): “*a directory, catalog, or folder, is an entity in a file system which contains a group of files and other directories*”. This metaphor was not without merit since, as stated in [4], “*The desktop and file & folder metaphor were created so that users could relate their computer-based systems to the paper-based systems they were used to*”. Unfortunately, a heavy heritage of this definition is the hierarchy that is imposed on the structure as well as the “*single-inheritance*” for each file: a file can be in one place at a time, like a sheet of paper can be in just in a folder a time.

The use of metaphor helped the diffusion of computers in the 80s when the personal computing entered even in the smallest office. With the evolution from the command line interfaces (CLIs) toward graphical user interfaces (GUIs) the office metaphor has been transposed in the images given to icons indicating folders, trash can and generic files (see Figure 1). Icons evolved in in term of number of colors or resolution, but the semantic of their use never changed at all.

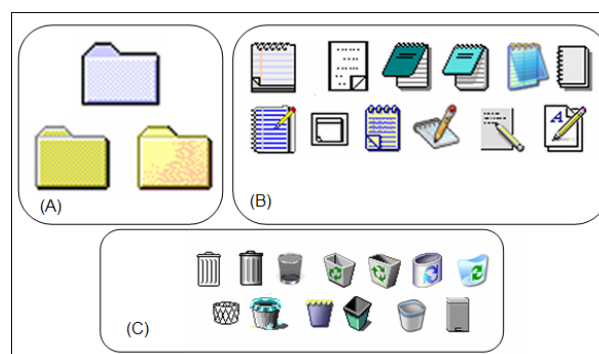


Fig. 1. The evolution of GUI through icons: group (A) folder icons, group (B) text file icons, group (C) trash can icons.

When the HFS idea arose the CPU power was a serious issue so, complex

operations on files like managing meta-data or complex directory structures, were nearly impossible. Another element to be considered in understanding the single inheritance limitation is the fact that in the 70s the amount of files managed was extremely small. In recent years mass storage available space has increased by a range of factors ([5]) and the number of files has done so proportionally.

Organization of the paper

In the next section, we motivate our work by shortly describing the structure and the definition of the hierarchical file system and its limitations. Then, we propose and analyze the “virtual directory” solution and we provide a short essay of various existing systems. We, then, introduce VENNFS the tool we are developing and some considerations about the undergoing development. The purpose of VENNFS feature requires knowledge of the Euler diagram in order to correctly create the directory structures. Therefore, we introduce a data structure, called *EulerTree*, that is designed for answering queries on Euler diagrams. The data structure is derived by the R-Tree (see [1]), a data structure that is able to answer *range queries* over a family of shapes in the 2-dimensional space.

2 Information organization

The hierarchical file system (HFS) structure [6], is the inspiring pattern for plenty of different implementations of file systems; it is well accepted and rarely put into discussions [7]. The hierarchy structure is appealing for various reasons:

- **Easiness of comprehension:** a hierarchy is easy to explain and easy to understand. In particular in a traditional office scenario the HFS can be put in direct relation with real world things. This is the reason for all the names used even today.
- **Great variety of visualization:** there are plenty of ways for visualizing a hierarchy data structure: the 2D node-link diagram [8], the horizontal family tree diagram [9] and the radial tree diagram [10]. Within the last decade, novel visualization methods have been developed for displaying large hierarchies, including the Treemap [8], cone tree [11], disc tree [12], hyperbolic tree [10], and 3D hyperbolic tree [13] visualizations.

Hierarchies are so common in modern operating systems, there exists a standard visual control called “tree-view” (see Figure 2) that is ubiquitous and whose use is intuitive and needs no explanation.

- **Hierarchy navigation** Another benefit of the hierarchical structures is the easiness of navigation and figuring out where the user is. This fact is particularly true considering how, traditionally, file systems have been browsed since their invention: using *Command Line Interface* (CLI).

In Figure 2 is shown the well known mechanism of current directory and the functionalities of command `cd`. The `cd` command (current directory or change directory in some implementations) and the symbol `..` (parent directory) allow to comfortably browse the hierarchy, with `cd ..` is possible to go up of a level and with `cd dirname` is possible to go down in the structure in the directory `dirname`.

To keep track of the *position* there is a special variable updated at every command called *path*: it states in which part of the hierarchy is the current directory, in a unambiguous way. The great advantage of this mechanism is that the path variable clearly states where the two possible directions, downward the hierarchy and upward will take the user.

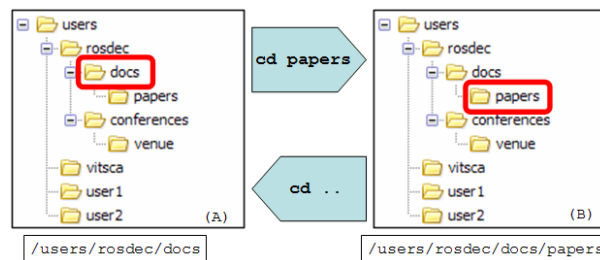


Fig. 2. The `cd` (current directory) command, it allows to browse the file system giving the exact position, the *path*, where the user *currently is*.

The evolution from Command Line Interface to the *Graphical User Interface* (GUI) simply gives the same functionalities of the `cd` command, but while in the CLI the user can be just in *one* point of the hierarchy, in the GUI a *multiple access points* mechanism is provided through the use of multiple windows. Indeed every window is an access point to the file system and has associated on it a path, of course this allows to expedite a lot of file management tasks, as shown in Figure 3.

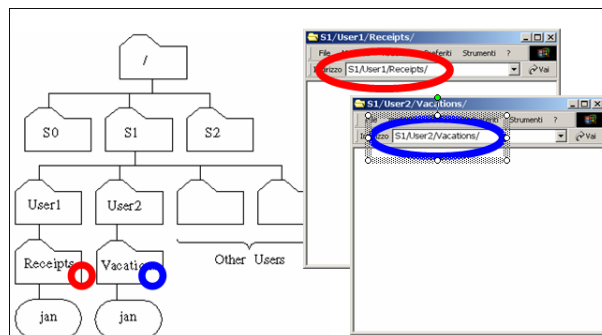


Fig. 3. The improvement took by the use of a *Graphical User Interface* (GUI): the access to the file system can be simultaneous in more current directories. This greatly helps file manipulations.

Hierarchies have been so widespread since the advent of personal computing they are always considered “*the natural way to organize things*” even in

scenarios not directly file related , for instance our bookmarks, our emails, our music etc. . . .

The limitation of using hierarchies as the unique way to categorise things is that in a rapidly evolving technological scenario some aspects are deliberately ignored or not exploited enough.

- CPU power increased dramatically. We can do more with our computer, like better and efficient retrieval “on-the-fly”, supportive organization of files and documents.
- Mass storage increased in size and speed. The more space available, the more files the user memorizes, and the bigger these files are, often meaning that each file is related to various different subjects.
- General items, so not “just files”, are usually kept on our disks and these items can be very informational rich like music, for instance, with some expressive associated meta-data about authors, genre, year of production etc. . . . In particular for music (e.g. .mp3 files) photos, etc. . . , the meta-data are really reliable because they are generated in an automatic way.

Organizing information in a hierarchy is so widely accepted that even applications designed in recent years use the same hierarchical organization like directories on a hard disk.

A clear example of this trend is the organization of Internet bookmarks in our browser (e.g. Internet Explorer, Mozilla Firefox, Opera, etc. . .).

Bookmarks refers to websites which are, of course, full of information that could be used. For instance, this information could be used to regroup bookmarks by topic in a clever way, or they could be put in relation to each other, or could be automatically classified into categories.

But what actually is implemented in browser is a hierarchical organization of bookmarks managed as they were unexpressive files.

The limitation of hierarchies is the *single inheritance*: an item can be just in one leaves of a tree structure. This limitation also is clearly stated in a paper [14], where a study on a real office was undertaken in order to focus the issues in the daily work. In particular is clearly emphasized that “*Simply allowing the same document to be easily put in several categories is one way computers can simplify classification*”.

3 Virtual Directories

Where the limitation is the single-inheritance to manage the multiple natures of files, the solution can be the *multiple-inheritance*.

Multiple-inheritance, in the most intuitive sense, is strongly connected to the nature of resources currently managed by our PCs.

Indeed the more space is available on our hard disks the more files are available, the more the file complexity and the heterogeneity of data increases.

3.1 Symbolic links

A brief consideration is needed about *symbolic links* offered by file systems to let a file “appear” in two or more positions in the hierarchical directories structure.

A symbolic link is a particular file whose behavior is similar to a *pointer*, when a command is performed on a symbolic link what really happens is that this command is performed on the file pointed by it. Symbolic links are quite powerful, they can even make remote resources appear to be local.

The weakness in the use of symbolic links is that they are prone to some inconsistencies, like when the pointed file is deleted or moved. Symbolic links can be suitable and even elegant, for the single file case.

3.2 Definition

A *Virtual Directory* (VD) is a directory that can be accessed in the usual way, but which is computed on demand answering to a query. From a “traditional file system” view, a virtual directory is an aggregation of files created at the application level, while usual directories are real structures written in the physical sectors of a disk.

Virtual directories are different from a search tool, a search tool *finds* files matching the rules that users provides, while a virtual directory *retrieves* files continuously and provides them through a traditional directory interface. Being virtual directories assembled on demand, the presence of a file in one virtual directory does not mean that this file will not be present in others virtual directories: a solution to the single-inheritance.

Of course the virtual directories computation usually ignore preexistent hierarchies in which files are kept, the traditional file system becomes *flat*.

3.3 Implementations

The need for a systematic, well integrated solution to single inheritance, is felt by various manufacturers, which intend to provide users system-wide solutions to categorize files in different manner than a hierarchy.

We provide here a small assay of the various technology will be proposed in these years.

Semantic File System. In [15] a “Semantic File System” is described in which files are kept in a traditional file system but are accessed using the virtual directory mechanism.

The files which are managed in the system described in the paper are news from the USENET archive, which are very “rich”, allowing comparison between items.

Presto. In the system proposed in [16] each resource has attached to it a variable number of user defined attributes. Once again a virtual directory is created on demand by user queries. Presto uses a data base to keep infor-

mation about files managed. Presto system enables user to define whatever attribute name for files, giving great flexibility to the kind of classification.

Some interesting points in Presto are (a) the attributes are kept in an data base, (b) Presto provides access to resources through a NFS (Network File System) interface, achieving, in this way, a strong compatibility with existing applications.

WinFS. The yet to come Microsoft *WinFS* (Windows Future Storage) is one of the pillars of the new OS codenamed Longhorn. WinFS ([17]) will provide a robust full fledged data base as undergoing flat file system.

Also in WinFS the file system is accessed assembling virtual directories. Widespread traditional directories like “My Music” and “My Pictures” are actually yielded through predefined virtual directories.

Instead of a HFS to organize information, WinFS uses a *direct acyclic graph* of items (DAG). It is a set of stored items and their relationships whose physical storage is a relational database providing support to store any item hierarchy. In the intents of the developers, with this novel storage method WinFS will offer search capacities never dreamed of before in file systems. It is possible to find items according to the value of their properties and even to the value of the properties of items related to them.

One of the problem the developers are facing is the computational cost of running a full fledge data base, called *Yukon*, to provides the relational functionalities needed.

Once again in [14] asserts how the automatic classification would greatly improve work, at the cost of some user’s effort to keep meta-data consistent and reliable.

3.4 Search tools

There is another category of tools that are related to the file system structure, actually considering it *absolutely unstructured*. These tools have the general appearance of a gadget embedded somewhere in the usual operating system user interface.

Storage ([18]) intends to replace the traditional file system with a new *document store* in which documents are kept unstructured and whose retrieving is provide through *natural language queries*.

Queries results can be kept in virtual directories backward compatible with existing, traditional, file systems and applications.

Spotlight ([19]) is a searching tool embedded inside the Tiger release of the Mac OS X. Spotlight takes the decision of categorizing just certain type of documents and media.

Spotlight works like a search boxes which provides results in per-media categorized flavor (see Fig. 4.(b)). This approach also is matched by the common search box present in Microsoft Windows (see Fig. 4.(c) for the Longhorn version). Spotlight also allows to save the search results in virtual

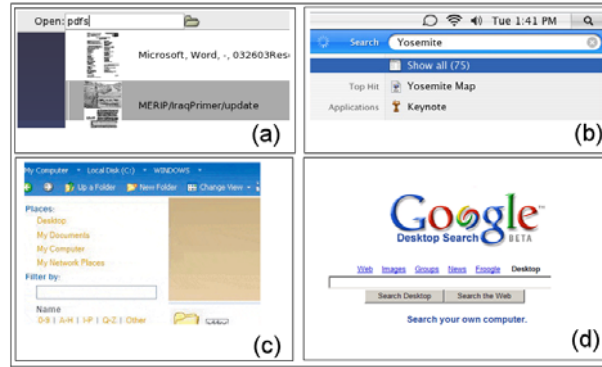


Fig. 4. Search boxes embedded in operating system. They categorizes almost all files allowing system wide searches. (a) Storage, (b) Apple: Spotlight (c) Longhorn: Search box (d) Google Desktop

directories.

Google Desktop ([20]) is a tool provided by Google in order to extend the search result of the well-known on-line search engine with results yet present on the hard disk of the user.

Google Desktop continuously scan the hard disk in order to find certain files it can handle, mainly textual file like `.doc`, `.pdf`, `.html` etc...

Some ad hoc solutions have been proposed also for delimited context like email clients:

Ximian Evolution [21] has *vFolders*. A Ximian Evolution vFolder looks like a folder but has no messages physically attached to it. Instead, a vFolder is defined by a set of criteria, like a message search. The does not have to manually enter the search criteria every time: a vFolder always contains the latest messages in all physical mail folders that match its criteria.

The Bat! [22] has virtual folders too that automatically collect all emails matching fine-grained criteria that works in the usual virtual directories manner.

4 Euler Diagrams for Virtual Directories

The application that we present here is a tool that can help the user during daily activities. Our work follows the research line that started by developing VENNFS [23] that allowed users to place documents and categories on a plane where files may belong to multiple categories at once, by using well-known and intuitive Euler diagrams to represent graphically each category. We placed particular care in designing an interface that, though fully visual, asks only few and quick interactions by the user.

4.1 VENNFS *main features*

We provide here a short survey of the current version of VENNFS presents a virtually infinite surface on which users can freely draw rectangles indicating a set. Rectangles can be drawn in a very intuitive manner, the first click places the first corner of surrounding rectangle, then the shape can be freely deformed by moving the mouse. This easy mechanism allows the user to figure out what is about to be created. The interesting point in the creation of a new set is that while deforming the shape the entire diagram is updated because it is important to allow user to figure out which are the newly created intersections.

A similar mechanism of drawing categories in which put elements can be also found in [24], where is described MediaFinder a tool useful to automatically categorize media.

Several advantages can be found on the representation of files on the plane [25], since we push the user to set data relationships as spatial relations by representing closeness of topic (between categories as well as between documents) by means of proximity in the plane. The user is suggested, implicitly, to draw related categories close to each other, since it allows partial overlap of them, while totally unrelated categories are intuitively placed far away.

An interesting visual consequence of using the distance to relate documents is that filtering by topic is implicitly obtained by zooming on the region and zooming in/out to get at the desired level.

Our objective is that the user is given an instrument to easily draw the environment represented by the (unstructured) corpus of his/her own documents, place documents into (possibly multiple) categories and relate categories by proximity: the task of information retrieval for the user is made easier by building a cognitive map [26] of this environment. Since the environment is created by the user himself, the internalized analogy in the human mind of the physical layout (created by using our tool) becomes easier to grasp.

In a sense, one may see VENNFS as a way to make a cognitive map of its documents explicit and comfortably navigable.

In order to facilitate the navigation we provide the capability to place landmarks, since it is well known that their identification helps in navigating [27] as well as learning and memorizing [28]. Landmarks do not correspond to files. A list of them is shown to the user and each one can be easily reached by double-clicking on landmark's label. The use of landmarks has been studied also in [25].

An indication of how recent is the file is shown by using an easily interpretable metaphor: recent (i.e. recently accessed) files are “hot” (i.e. red) while old files are “cold” (i.e. blue) [14] with intermediate colors to represent intermediate age. Then, it is important to allow filtering over the time (by using a slider) in such a way to show only the files whose last modification date is below a given date.

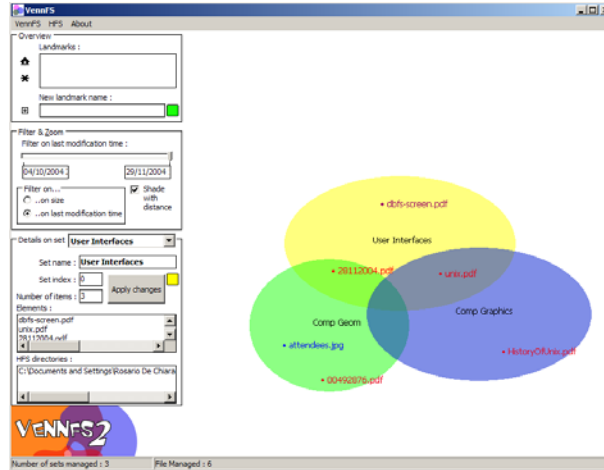


Fig. 5. A screenshot from the current version of VENNFS. The categories are yet visualized using ellipses.

4.2 Virtual Directories in VENNFS

VENNFS has been designed to effectively convey the structure that is kept in user’s mind onto a collection of files, in such a way that it can be stored, retrieved and updated. There is an implicit limit that all the automatic tools are facing: the connections between different files are often only available to human beings (i.e. “Two papers that were produced in the same physical place, during two different visits to a research center”) VENNFS does not intend to be an automatized tool for classifying documents and neither a file system (no matter of the name). Euler diagrams are extremely intuitive and particularly useful for the usage by a non professional user (see, e.g., how Euler diagrams are used for representing the results of complex queries in [29]).

The persistent nature of a Virtual Directory makes it a versatile tool since additional items can be (manually) added. Moreover, the query is constantly monitored and, therefore, changes occurring in the files present in the virtual directory (such as new entries) can be added. Careful tuning of the size of the data monitored as well as the frequency of monitoring is important to keep efficiency in the application (see Figure 6).

When new items are found, of course the user must be informed and the resources are placed in the plane in a “temporary” set of uncategorized items, see also [14] for deferred classification. By drag-and-drop, the user can then place the item in right place, as well as create new categories. Items that do not belong to the query results anymore (moved or deleted) are shown (for a short period of time) as slowly disappearing from the diagram in order to inform the user that a change occurred.

Another stimulating vision is that we can recur and use queries on Virtual Directories, whose result are Virtual Directories as well. It means that the user is given the opportunity to join together several virtual directories that represent filtering on different dimensions of the same material.

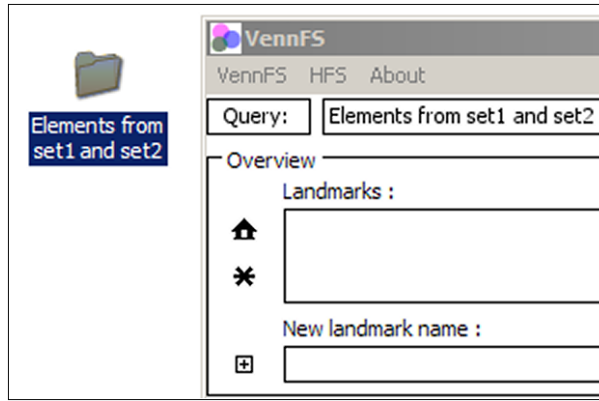


Fig. 6. An example of virtual directory created from a query.

4.3 Presentation to the user

Current graphics boards offer capabilities to create astounding graphics effects once relegated to special-purpose software (entertainment, CAD), and now available (for free) on our everyday personal computer. The trend is to exploit this hardware to greatly improve the appeal, the look but also the functionalities of traditional tools like file browser. This general trend is remarked also in [17].

In VENNFS we exploited such features (alpha blending, scaling, camera effects, smooth transitions etc...) so that the interface offered is more dynamic and supportive toward the users. As an example we can cite “smooth zooming” effect used to change context (switching among categories) without disorienting the user.

Also “category selection” is visualized by using a smooth animation (offered by the graphics board). Exploiting graphics hardware also allows to solve classical geometrical problems, for instance it can be used to rapidly answer queries like “in which polygons lie a point”.

In current development of VENNFS we are aiming to three main targets, as stated in [14]:

- **Creating Classifications:** this task is accomplished using the intuitiveness of Euler diagrams drawn by hand by users. Sets can be subjects, topics, directories, etc... and all the various aspect our life can be related to, and this concept is so easy to be understood that it needs no explanation even to novice users.
- **Classifying Information:** this is performed using the well know “drag’n’drop” mechanism which the user is well acquainted with. Dragging a file to a point in the plane of VENNFS intuitively “puts” the file in the sets overlapping in such point. The overlapping sets are rendered blending their colors. There is a stimulating view we are still investigating and is to associate to every category a set of properties that gets inherited by elements inside the set. This is the tempting way already crossed by others ([17], [16]) in which

documents and files are handled as rows in a data base. We also intend to provide significance to a drag'n'drop performed on a directory icon, this would create immediately a new set named after the directory.

During some early user test we got the suggestion of providing a sort of “*wizard*” that directly migrate an entire subtree from HFS to the diagram. This would be a good starting point to use VENNFS.

- **Retrieving Information:** this is an interesting functionality which involves the implementation of a boolean expression parser to perform queries on the diagram. The challenging part is to visualize the results of the queries, more than providing them in a textual manner. The current work in this direction intend to exploit once again the smooth zoom operation provided by OpenGL: the idea is to zoom out the diagram shading to gray tones and letting the results blink. The result is shown in Figure 7. Once visualized the query can be dragged somewhere onto the HFS in order to create a virtual directory.

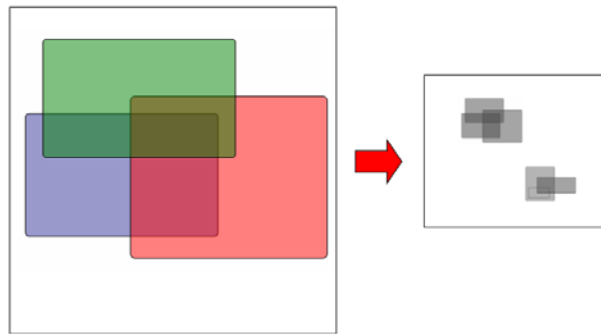


Fig. 7. Query result visualization: Elements from Red and LightBlue are shown zooming out to show the entire result and fading out sets color in order to better show the selected elements blinking.

5 A data structure for Euler Diagrams

In this section, we define a data structure, called *EulerTree*, that is going to be used by VENNFS in order to efficiently answer queries on Euler diagrams. The data structure is derived from the R-Tree (see [1]), a data structure that is able to support *range queries* over a family of shapes in the 2-dimensional space.

5.1 Definitions

We aim to manage categories (sets). A 2-dimensional space will be considered. Sets are well defined shapes (no free-hand set). In particular we restrict our attention to rectangular sets. This is not a real limitation to the aesthetics of the diagrams, as shown in [30]. We consider *set* and *rectangle* as interchange-

able terms. Of course sets can be wherever in the space, properly intersecting each other or one inside the other.

We provide here some informal definitions:

Space: We will refer to the 2-dimensional space. Actually the proposed data structure can be extended to other higher dimensional spaces but since our research is planned to be used in an experimental scenario we will not investigate further.

Set or Rectangle: In this paper we define *set* as rectangular axis aligned figures (isothetic rectangles).

Diagram: Let *diagram* denote a collection $C = \{C_1, C_2, \dots, C_n\}$ of n freely placed sets in the space. “Freely placed” means that set can be disjoint, overlapping or on inside the other. Every point in the space can be in $int(C_i)$ (the interior of C_i) or in $ext(C_i)$ (the exterior of C_i). The border of every set is in $int(C_i)$.

Subface: A *subface* is a region of plane contained by one or more set. In Figure 9.(a) a subface is grayed.

Subrectangle: A rectangular subface is called a *subrectangle*. Subrectangles are generated by overlapping sets of the diagram. In Figure 10 we show the result of SweepLine Algorithm (see Section 5.4) which cut a subface in one or more subrectangles.

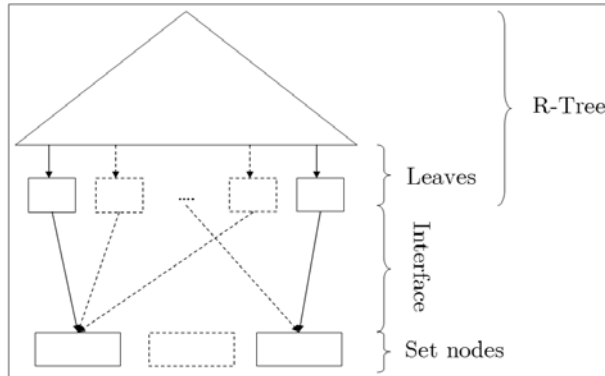


Fig. 8. The general aspect of the EulerTree data structure.

The EulerTree is based upon the R-Tree data structure so it implements all the queries the R-Tree data structure provides. R-Trees are used in various geometric contexts. The idea of using a more supportive data structure is based on the need for some diagram design hints, indeed the application that will use this data structure is VENNFS (see [23], [31])

In VENNFS we are also interested in a method to obtain diagram information referring to more than one set.

5.2 R-Tree description

The R-Tree (originally introduced by [32]) is designed to answer *range queries* (*window queries*): given a collection of objects in \mathbb{R}^d , a range query reports

all objects that intersect a d -dimensional axis-aligned query window, that is, a d -dimensional box. We will refer to the bidimensional space so d will be 2 and the objects will be isothetic rectangles. An R-Tree is a particular B -tree in which every leaf keeps an input box. Further details can be found in [1].

Given S a collection of n possibly intersecting boxes in \mathbb{R}^2 , a perfectly balanced box-tree for S can be built in $O(n \log n)$ time. A range query can be answered in $O(\sqrt{n} + k \log n)$, with k number of rectangles returned (in the bidimensional space).

5.3 The EulerTree data structure

We present a preliminary analysis of a data structure that appears to be suitable for efficiently representing Euler diagrams.

The EulerTree is an extension of the R-Tree designed to trace back the subfaces created by intersecting sets in the diagram. Figure 8.(a) describes how the R-Tree is expanded: a level of nodes, called *Set Nodes*, is added under the tree, it will keep a node for every set in the diagram.

The links between the R-Tree leaves and the Set Nodes are called the *Interface*: a link is created in the Interface between a Set Node and a subface leaf of the R-Tree if and only if the subface belongs to the set.

The data structure EulerTree intend to be used to retrieve information about a diagram that will change dynamically. The queries an EulerTree has to answer are:

- Which are the proper subsets of a set A ?
- Which are the sets that intersect a set A ?

The diagram will be drawn one set at a time. Once inserted a set will probably overlap other sets generating one or more intersection with the associated subfaces. This modification will be immediately reflected on the EulerTree.

Figure 9.(a) shows an Euler diagrams made of two intersecting sets: sub-rectangles s_1 , s_2 and s_4 are created splitting the subface grayed. Figure 9.(b) the EulerTree relative to the diagram.

The insertion of a new set in the diagram is performed using the query and delete operations provided by R-Tree, in two phases:

- (i) In the first phase we query the R-Tree to check which are the subfaces involved in the insertion of the new set.
- (ii) In the second phase these rectangles are deleted from the R-Tree, handled and re-inserted in the R-Tree.

To better explain the details we give a brief description of what an R-Tree is.

The EulerTree is updated by Algorithm 1 that uses the Algorithm SweepLine-Split described in the next subsection.

Let R denote the rectangle (set) that we would like to insert into the EulerTree. First we add it to the tree-structure and make a query in EulerTree

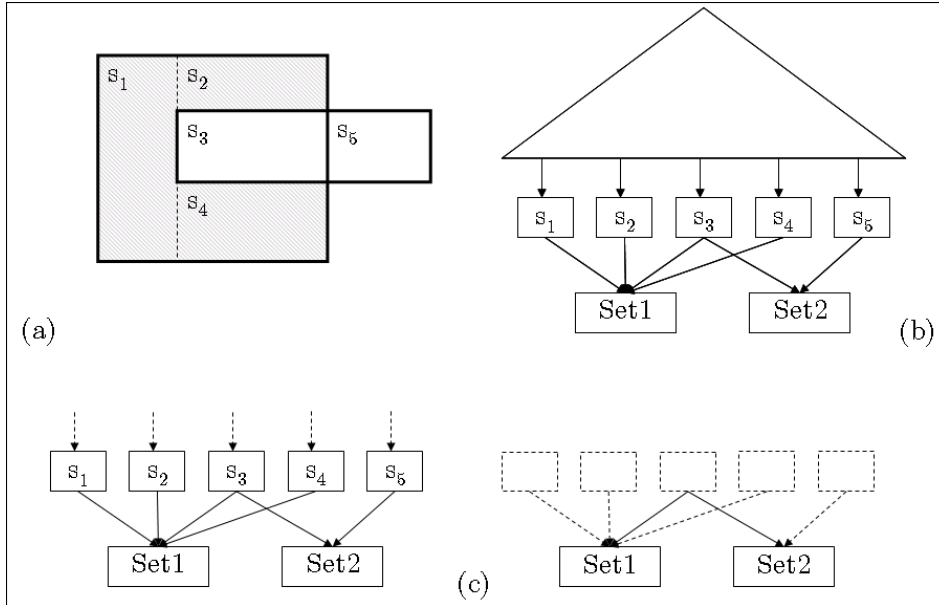


Fig. 9. (a) An Euler diagram and the associated EulerTree (b), in gray a subspace. In (c) the detail of the Interface and how the intersections are managed. In particular the subrectangle s_3 belongs to both sets and this situation is correctly reported by edges.

Algorithm 1 Insert a new rectangle in the data structure

Input: R the new rectangle to insert in T , the EulerTree.

Output: Updated T

- 1: Add a Set Node R to EulerTree.
 - 2: $s := \langle r_1, \dots, r_k \rangle \leftarrow \text{Query}(T, R)$.
 - 3: Remove s from the R-Tree of T .
 - 4: $z := \langle \rho_1, \dots, \rho_m \rangle \leftarrow \text{SweepLineSplit}(R, s)$.
 - 5: Insert z in T connecting every s_i to the right Set.
-

to obtain all subrectangles that intersect R .

These subrectangles are deleted from the R-Tree.

Next we run the SweepLineSplit algorithm to compute the new set of subrectangles. Each subrectangular face in the resulting subdivision contains pointers to the set nodes it belongs to. Hence, adding these faces to the R-Tree updates the EulerTree. Note that each rectangle in the R-tree is assumed to keep an attribute that is a pointer list of all set nodes it belongs to.

5.4 Sweepline split algorithm

In order to calculate the plane subdivision obtained by the insertion of a new rectangle in the diagram we use a well known technique, called *map overlay*.

The rectangles are represented by a doubly-connected edge list.

This data structure represents planar subdivisions, and consists of a set of vertices, a set of half-edges and a set of faces, all connected according

to their adjacency in the planar subdivision. A half-edge is a directed edge representing one side of a line segment. Thus, a line segment is represented by two half-edges of opposite directions.

The doubly-connected edge list representing our rectangle subdivision has one unbounded face being part of none of the Set Nodes. This face needs not be a rectangle in itself. All other faces must be rectangle shaped and they represent the subrectangles that in turn constitute the intersection of the Set Nodes. Hence, each such face has a list of pointers keeping track of what Set Nodes it belongs to.

When we insert a new rectangle, the set of subrectangles needs to remain disjoint after the operation. Hence, it is necessary to split the intersected rectangles and the query rectangle into new subrectangles. This is done using the sweepline technique. The input of the splitting algorithm is two subdivisions representing the query rectangle and the set of rectangles intersecting it, respectively. The output is the new set of disjoint subrectangles.

The algorithm works in two phases. In the first phase we compute the overlay of the two subdivisions. The overlay of two subdivisions S_1 and S_2 is defined as the planar subdivision induced by the edges from S_1 and S_2 . Each face in the overlay keeps pointers to the faces in S_1 and S_2 that contain it. Hence, after computing the overlay it is a simple task to update the Set Node pointer list of each face in the overlay.

The overlay is computed using the MapOverlay algorithm in Chapter 2.3 of [33]. The time complexity of this algorithm is $O(k \log k + k' \log k)$, where k is the number of rectangles intersecting the query rectangle and k' is the number of faces in the overlay.

The overlay contains faces which are not rectangles. These must be partitioned further. We use a simple sweepline algorithm to find such a partition.

It works as follows: the sweepline goes from left to right. At all times we impose that no vertical half-line to the left of the sweepline bounds a bounded face and has a reflex vertex as either origin or destination. If no bounded face contains reflex vertices they are all convex which in our case implies that they are rectangular. If we find a reflex vertex, we add a vertical edge, i.e., a twin of half-edges, to the edge list making the vertex non-reflex. All updates comprise a subset of those done in the MapOverlay algorithm.

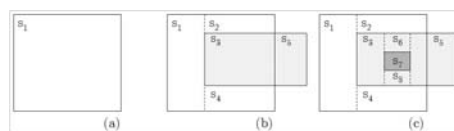


Fig. 10. An example of the application of the SweepLine Algorithm.

We note that adding a vertical edge splits a face in two parts. However, the face attribute remains the same, since the new rectangles belong to the same Set Nodes. The time complexity of the second phase is $O(k' \log k')$.

5.5 Deleting a set

To remove a given set from the EulerTree we provide Algorithm 2.

Algorithm 2 Remove a set from the data structure

Input: T the EulerTree and S a set to remove from T .

Output: Updated T

- 1: **for** all subspaces r incident on S **do**
 - 2: **if** $out_degree(r) == 1$ **then**
 - 3: Remove r from the R-Tree of T
 - 4: Remove the edge $\langle r, S \rangle$ from the Interface
 - 5: **end if**
 - 6: **end for**
 - 7: Remove S from the EulerTree T
-

The deletion of set S , is performed by simply removing from the R-Tree all the subspaces of S which are not shared with other sets in the diagram. For a given subrectangle, this condition is verified, inside the for..loop in Algorithm 2, just checking that the out degree is exactly 1 meaning that the subrectangle is defined by set S . An example is given in Figure 11.

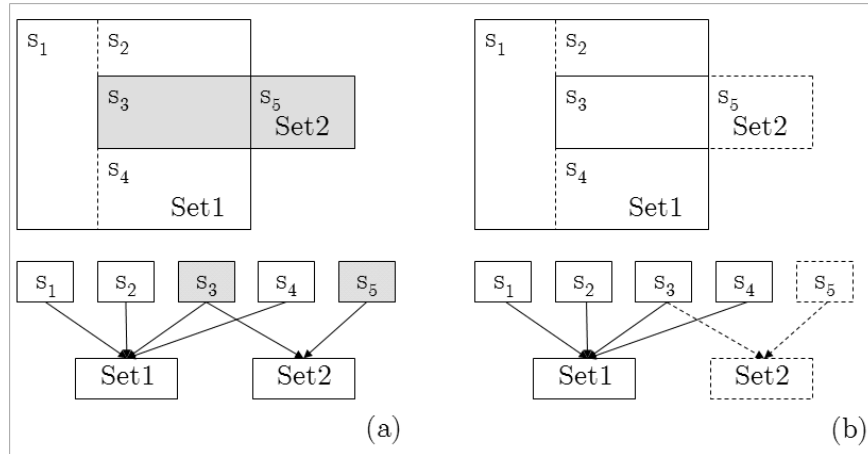


Fig. 11. An example showing the delete operation applied on *Set2*. The effects of the operation are displayed both in the diagram as well as in the interface of the EulerTree. the diagram and just the Interface of the EulerTree. In (a) the *Set2* and the two subrectangles, s_3 and s_5 , checked in the for cycle of Algorithm 2 are grayed. (b) shows the result of the deletion of *Set2*, just the subrectangle s_5 is actually removed, while the subrectangle s_3 is kept because it is shared with *Set1*.

A serious drawback of the deletion algorithm is the presence of some unnecessary subrectangles (for instance see s_1, s_2, s_3, s_4 in Figure 11) that could be merged in bigger rectangle

6 Conclusions and future works

The experience we are having with VENNFS provides us good feedbacks from users, but of course this is not a serious usability test to which every tool who intend to provide new interactions must be submitted to.

Another step we intend to perform is to use Euler diagrams to organize information in other contexts like bookmarks and emails, for instance. Managing emails and bookmarks would be different from managing generic files since, for instance, there exist easy ways of comparing two emails (e.g. words in the subject, recipient address etc. . .), and these comparison methods could be used to automatically place elements in sets.

A challenging goal to meet is the design of a compact representation of a diagram that can compete with the well-known hierarchical representation. For hierarchies, as we stated, there are plenty of different and efficient visualization techniques, but for daily work the most spread and effective is the tree view (as shown in Figure 2). To allow a diffusion of Euler diagrams in common use applications, a compact, easy to explain, like the “tree view” representation, is needed, in order, for instance, to replace the use of hierarchies in managing internet bookmarks.

On conclusion some critics may arise to the intrinsic limitations to the expressiveness because of using isothetic rectangles: for instance, in a 4 sets diagram could be impossible to render certain intersections or wrong intersections could be inferred (see [?]). Using more complex figures (e.g. k-gons and blobs) would go round these limitations.

Being our research strongly application-oriented, we prefer the easiness of managing (e.g. drawing, stretching, moving) rectangular sets, instead of potential expressiveness of diagrams designed using more complex figures.

References

- [1] Pankaj K. Agarwal and Mark de Berg and Joachim Gudmundsson and Mikael Hammar and Herman J. Haverkort, “Box-trees and R-trees with near-optimal query time,” in *Symposium on Computational Geometry*, 2001, pp. 124–133.
- [2] D. Gentner and J. Nielsen, *The Anti-Mac Interface*. [Online]. Available: <http://www.acm.org/pubs/cacm/AUG96/antimac.htm>
- [3] “Wikipedia: Directory.” [Online]. Available: <http://en.wikipedia.org/wiki/Directory>
- [4] S. Fertig, E. Freeman, and D. Gelernter, “Finding and reminding reconsidered,” *SIGCHI Bull.*, vol. 28, no. 1, pp. 66–69, 1996.
- [5] H. R. Grochowski E., “Future trends in hard disk drives,” *IEEE Transactions on Magnetics*, vol. Vol.32, Iss.3, pp. 1850–1854, 1996.

- [6] D. M. Ritchie, “The evolution of the UNIX time-sharing system,” *BSTJ*, vol. 63, 8, pp. 1577–1594, 1984. [Online]. Available: citeseer.ist.psu.edu/ritchie84evolution.html
- [7] “The naming system venture.” [Online]. Available: <http://namesys.com/whitepaper.html>
- [8] B. Johnson, “TreeViz: treemap visualization of hierarchically structured information,” in *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM Press, 1992, pp. 369–370.
- [9] D. E. Knuth, *Fundamental Algorithms*, 3rd ed., ser. The Art of Computer Programming. Addison-Wesley, 1997, vol. 1, pp. 308–316.
- [10] J. Lamping and R. Rao, “Laying out and visualizing large trees using a hyperbolic space,” in *ACM Symposium on User Interface Software and Technology*, 1994, pp. 13–14.
- [11] G. G. Robertson and J. D. Mackinlay and S. K. Card, “Cone Trees: Animated 3D Visualizations of Hierarchical Information,” in *Proc. of CHI-91*, New Orleans, LA, 1991, pp. 189–194.
- [12] C.-S. Jeong and A. Pang, “Reconfigurable disc trees for visualizing large hierarchical information space,” in *Proceedings of the 1998 IEEE Symposium on Information Visualization*. IEEE Computer Society, 1998, pp. 19–25.
- [13] Tamara Munzner, “Exploring Large Graphs in 3D Hyperbolic Space,” *IEEE Comput. Graph. Appl.*, vol. 18, no. 4, pp. 18–23, 1998.
- [14] Thomas W. Malone, “How do people organize their desks?: Implications for the design of office information systems,” *ACM Trans. Inf. Syst.*, vol. 1, no. 1, pp. 99–112, 1983.
- [15] J. W. O’Toole and D. K. Gifford, “Names should mean what, not where,” in *Proceedings of the 5th workshop on ACM SIGOPS European workshop*, 1992, pp. 1–5.
- [16] P. Dourish, K. W. Edwards, A. LaMarca, and M. Salisbury, “Presto: an experimental architecture for fluid interactive document spaces,” *ACM Transactions on Computer-Human Interaction*, pp. 133–161, 1999.
- [17] “WinFS.” [Online]. Available: <http://msdn.microsoft.com/Longhorn>
- [18] “Gnome storage.” [Online]. Available: <http://www.gnome.org/~seth/storage/features.html>
- [19] “Spotlight technologies.” [Online]. Available: <http://www.apple.com/macosx/tiger/spotlight.html>
- [20] “Google desktop.” [Online]. Available: <http://desktop.google.com/>
- [21] “Evolution.” [Online]. Available: <http://gnome.org/projects/evolution>

- [22] “Ritlabs: The Bat.” [Online]. Available: <http://www.ritlabs.com/en/>
- [23] R. De Chiara, U. Erra, and V. Scarano, “VennFS: a Venn Diagram File Manager,” in *Proc. of the Seventh International Conference on Information Visualization, IV 2003, 16-18 July 2003, London, UK*, 2003.
- [24] Hyunmo Kang and Ben Shneiderman, “MediaFinder: an interface for dynamic personal media management with semantic regions,” in *CHI '03 extended abstracts on Human factors in computing systems*. ACM Press, 2003, pp. 764–765.
- [25] G. G. Robertson, M. Czerwinski, K. Larson, D. C. Robbins, D. Thiel, and M. van Dantzich, “Data mountain: Using spatial memory for document management,” in *ACM Symposium on User Interface Software and Technology*, 1998, pp. 153–162.
- [26] E. C. Tolman, “Cognitive maps in rats and men,” *Psychological Review*, vol. 55, pp. 189–208, 1948.
- [27] A. Dillon, J. Richardson, and C. McKnight, “Navigation in hypertext: a critical review of the concept,” in *Diaper, D., Gilmore, D., Cockton, G., and Shackel, B. (Eds) INTERACT '90*. North Holland, Amsterdam, 1990.
- [28] C. Johns and E. Blake, “Cognitive maps in virtual environments: Facilitation of learning through the use of innate spatial abilities,” in *Proc. of 1st International Conf. on Computer Graphics, Virtual Reality and Visualisation in Africa, Cape Town, South Africa. 2001*, 2001.
- [29] A. F. Blackwell, K. Marriott, and A. Shimojima, Eds., *Diagrammatic Representation and Inference, Third International Conference, Diagrams 2004, Cambridge, UK, March 22-24, 2004, Proceedings*, ser. Lecture Notes in Computer Science, vol. 2980. Springer, 2004.
- [30] J. Flower, P. Rodgers, and P. Mutton, “Layout metrics for euler diagrams,” in *Seventh International Conference on Information Visualization (IV03)*. IEEE, January 2003, pp. 272–280.
- [31] R. De Chiara, U. Erra, and V. Scarano, “A visual adaptive interface to file systems,” in *Proceedings of the working conference on Advanced visual interfaces*, 2004, pp. 366–369.
- [32] A. Guttman, “R-trees: a dynamic index structure for spatial searching,” in *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, 1984, pp. 47–57.
- [33] M. Hammar, M. de Berg, J. Gudmundsson, and M. H. Overmars, “On r-trees with low stabbing number,” in *European Symposium on Algorithms*, 2000, pp. 167–178. [Online]. Available: citeseer.ist.psu.edu/313683.html
- [34] “Limitations on diagrammatic representation and reasoning.” [Online]. Available: <http://plato.stanford.edu/entries/diagrams/#3.1>