Adaptive Layout for Interactive Documents

Dissertation
zur Erlangung des akademischen Grades

Doktor-Ingenieur (Dr.-Ing.)

der Fakultät für Informatik und Elektrotechnik
der Universität Rostock

vorgelegt von

Kamran Ali

geb. am 21. Juli 1977 in Layyah, Pakistan
aus Rostock

Rostock, 31. März 2009

urn:nbn:de:gbv:28-diss2009-0159-3

In the memory of my father Sh. Zafar Ali Pakistani (d. 1999),
upon whose strong shoulders I rode to school as a child

For my mother Aziz Begum,
whose love knows no bounds

Abstract

Digital documents pose many challenges to their effective presentation. Due to the dynamic nature of digital media, and to a wide variety of clients connected to different display devices, the decision of how to lay out the information tailored to the current scenario becomes a challenging task. In order to effectively address this issue, digital documents need automated processes to produce dynamic design solutions. This thesis presents a novel approach to create automated layouts that could adapt according to the screen size and contextual requirements. The focus of the research is to develop layout adaptation techniques for rich illustrative material which the users can interact with. The adaption techniques not only consider the global layout decisions for elements on the screen, but also deal with the content and layout adaptation of individual illustrations in the layout.

The dynamics of digital documents may prevent the designer from finishing templates for the compound content that is unknown at design time. The methods proposed here aim at creating optimal layouts of content even when the templates are partially designed and the varying amount of illustrated material needs to be fit to the layout. While the state of the art layout techniques have been influenced by the designer-specified constraints, our work takes into account both the design constraints and semantics of the content to make layout decisions. A unique integrated solution has been developed that combines constraint-based and force-directed techniques to create adaptive document layouts of different kinds—grid-based and non-grid layouts. The integrated approach allows the document to retain the overall design intended by the designer, and yet to make adequate changes to the layout to meet the constraints imposed by the content authors and viewers. The work also explores the authoring tools necessary for creating adaptive documents.

In order to enrich the digital documents with dynamic illustrations, novel annotation layouts are developed which adapt the annotated illustrations to match the contextual requirements over time. Based on the corpus analysis of hand-made illustrations, a variety of external and internal annotation styles are classified and implemented. The annotation layout algorithms work in image space which makes them equally suitable to annotate 2D and interactive 3D visualizations on the fly.

The early results of this work demonstrate its potential to contribute towards the improvement of dynamic design solutions in interactive documents.

Zusammenfassung

Die effektive Darstellung digitaler Dokumente stellt viele Anforderungen auf. Auf Grund der dynamischen Eigenschaften von digitalen Dokumenten und der großen Variation an Clients und mit ihnen verbundenen unterschiedlichen Ausgabegeräten, wird die Anpassung der Datendarstellung an das jeweilige Umgebungsszenario sehr anspruchsvoll. Um dieses Problem effektiv zu behandeln, benötigen digitale Dokumente automatisierte Verarbeitungsmethoden, die dynamische Designlösungen anbieten. Diese Dissertation stellt eine neue Herangehensweise für die Erstellung automatischer Layouts (Darstellungsverteilungen) dar, die sich an die Bildschirmgröße und den Kontext anpassen. Der Fokus des Forschungsvorhabens ist dabei die Entwicklung von Layout-Anpassungstechniken für umfangreiche illustrative Materialien, mit denen der Nutzer interagieren kann. Die Anpassungstechniken betrachten dabei nicht nur globale Layout-Entscheidungen bzgl. der Elemente am Bildschirm, sondern behandeln gleichermaßen den Inhalt und die Layout-Anpassung individueller Illustrationen im Layout.

Die dynamischen Eigenschaften digitaler Dokumente könnten den Designer davon abhalten, Dokumentvorlagen für die unterschiedlichen Inhalte (Text, Bild, Video, ...) bereitzustellen, die zum Zeitpunkt des Entwurfs der Dokumentvorlage noch nicht bekannt sind. Die hier entwickelten Methoden richten sich an die Erstellung eines optimalen Layouts, selbst wenn die Dokumentvorlagen nur zum Teil entworfen wurden und die unterschiedliche Menge an illustrativem Material an das Layout angepasst werden muss. Während aktuelle Layout-Techniken von den vom Designer spezifizierten Constraints (Einschränkungen) beeinflusst werden, werden in dieser Arbeit beide Aspekte, die während des Entwurfs spezifizierten Constraints sowie die Semantik des Inhalts, für die Layout-Entscheidungen berücksichtigt. Eine einheitliche Lösung wurde entwickelt, die constraint-basiert und force-basierte Techniken für die Erstellung von adaptivem Dokumenten-Layout verschiedener Art kombiniert—gitterbasiert und gitterlose Layouts. Der einheitliche Ansatz ermöglicht es, das vom Designer beabsichtigte allgemeine Design beizubehalten und gleichzeitig adäquate Änderungen vorzunehmen, um weiteren Einschränkungen zu entsprechen, die vom Autor des Inhalts wie auch vom Betrachter der Darstellung des Dokuments auferlegt werden. Diese Arbeit untersucht dabei ebenfalls die notwendigen Werkzeuge für die Erstellung adaptiver Dokumente.

Um die digitalen Dokumente mit dynamischen Illustrationen anzureichern, sind neuartige Annotationslayouts entwickelt worden. Diese passen die annotierten Illustrationen dy-

namisch an den Kontext an. Basierend auf der umfangreichen Analyse von Hand erstellter Illustrationen, wurde eine Vielfalt von externen wie internen Annotationsstilen klassifiziert und algorithmisch umgesetzt. Die Annotationslayoutalgorithmen arbeiten dynamisch im Bildraum, wodurch sie gleichermaßen für die Annotation von interaktiven 2D und 3D Visualisierungen geeignet sind.

Die ersten Resultate dieser Arbeit besitzen das Potential, für die Verbesserung dynamischer Designlösungen interaktiver Dokumente beizutragen.

Acknowledgements

In the Name of Allah, the Beneficent, the Merciful.

On this page, I would like to express my gratitude to all those who have helped me during my doctoral research, which was completed at the University of Rostock and the Otto–von–Guericke University of Magdeburg.

Firstly, I am very grateful to my supervisor Prof. Dr. Heidrun Schumann for assisting me in my research. Her constructive feedback and suggestions have always been very useful. This dissertation would not have been finished without her help. I would also like to say thanks to my former supervisor Prof. Dr. Thomas Strothotte, for his deep concern for me despite the change of his jobs, and for arranging the funding. Next, I thank my advisor and external reviewer Prof. Dr. Knut Hartmann for his inspiration, advice, and critical evaluation of my work. I have learnt a lot from him over the years. I would also like to thank my other external reviewer, Prof. Pavel Slavik for reviewing this thesis.

The computer graphics group at the University of Rostock has been a great place to work because of the many great collagues. In particular I would like to say thanks to Georg Fuchs for the fruitful discussions I had with him. I also thank Sebastian Schwanke for his last-minute critical review of the abstract.

I owe much thanks to Angela Brennecke. Angela and I have been through the same course during our research work. It is beyond the words to express the amount of motivation, help, and moral support I have received from her. Our Ph. D. slogan was *GBG (Go Baby Go!)* which did miracles at the end.

I am also very thankful to all my colleagues at the Otto–von–Guericke University Magdeburg, especially to Timo Götzelmann whom I have collaborated with during most of my research work in Magdeburg.

I have been so lucky to have few great friends who love me for who I am. Faizan Ahmad, Jens Grubert, Sowdagar Sadiq, Muhmmad Saqib, Shafiq Raja, Mohsin Raza, Mian Sohail Akram, Alvin, Mushtaq Alam, Oleg Nesterenko, and the lovely couple Anna Láng and Shabi Ulzama. Shabi did a wonderful job in proof-reading most of my thesis at a very short notice of time. He is DA man!!!

From Magdeburg, there are too many people to list them all here but I just want to thank them all especially Elli and Ayaz Farooq. My thanks also go to my friends in Rostock, in particular to Muhammad Adeel, Muhammad Nawaz and Muhammad Farooq.

Finally, my biggest thanks go to my mother Aziz Begum for her love, her patience and countless prayers for me throughout my life. My sisters and brothers, Salma, Zulfiqar, Seema, Shabnam, Sheeba, Saniya, and above all Zeeshan have been a constant source of support.

I miss you, Dad! Till we meet one day, I will always miss you.

Contents

List of Figures

Chapter 1

Introduction

Rapid growth of internet and advanced web technologies have seen large volumes of information being delivered over web. In the digital networked world, the content can be frequently updated by authors, information can be viewed at different display devices, individual clients can request information in different forms and set their own viewing preferences. Therefore, the dynamic requirements of the digital media restrict that the documents cannot be formatted in advance. A simple approach of using basic layout templates may result in static layout which is scrolled up and down within a window and is inflexible to changes in context. Hence, traditional processes of content creation, editing, and quality assurance have to be completely revised in dynamic environments. What is needed in digital media is that the presentations adapt themselves according to the dynamic changes in both the content and in response to individual users' needs.

1.1. Motivation and Problem Discussion

Figure 1.1 shows a scanned article from a Visual Dictionary book [Doc02]. A detailed review of the illustrated page reveals several aspects of design—title at top, thumbnail image, text description on left, main illustration in the center and several other related images placed around it. Moreover, the figures are richly annotated with text *labels* or

annotations.[1] This is a classic example of pictorial material laid out carefully by expert human illustrators. In traditional print media, the illustrated documents have been serving for a long time.

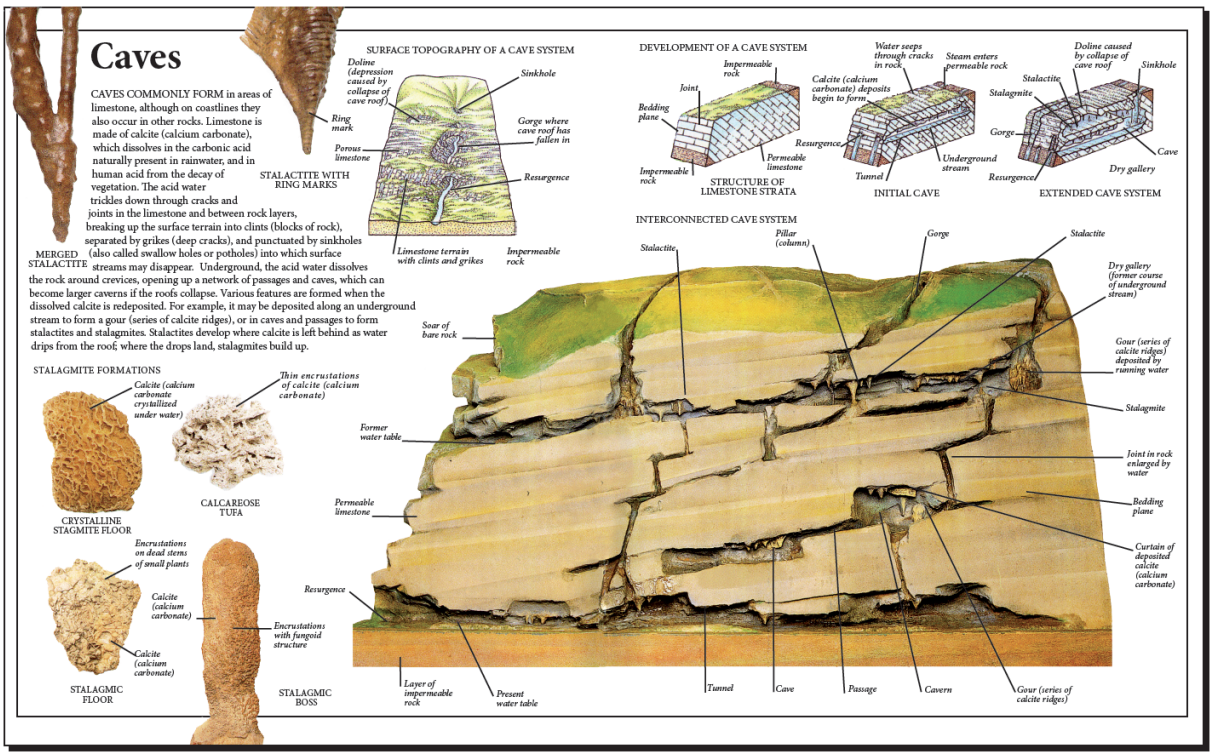


Figure 1.1.: An example from Visual Dictionary [Doc02].

The reuse of printed material (e.g., text books, encyclopedia, or technical documentation) in interactive applications is very promising due to the high quality of their content. However, presently, the web browsers provide limited functionality in adapting the layout of illustrated material and embed illustrations as static objects. The static illustrations in a digital document exhibit a lack of responsiveness to the changes in content, layout and user interaction. As a result, the static illustrations may not offer significant improvements in reading over their traditional counterparts. Besides, more often than not the information recipients are going to be many individual users who are anonymous and using different display devices. Variations of display resolution and computing power of the electronic devices may require that the document is presented differently for each reader. Otherwise, a static design of the document might result into a disaster. With the current available

1 In literature, the terms *label* and *annotation* are used interchangeably. Authors prefer one term or the other. Within the scope of this thesis, there is no distinction made between a label and an annotation. Two other terms *labeling* and *annotation layout* are also used vice versa.

technology, it is hard for the designers to specify how the layout should be adjusted under different scenarios. As a result, the viewers often experience static layouts that quickly become unsatisfactory when the content or device capabilities change.

The situation becomes even harder for layout with many illustrations. While web browsers guarantee the readability of text on all devices and offer at least some room to interact with text, illustrations are inflexible elements in the layout. *Interactive illustrations* would offer new possibilities for online documents without interfering with traditional editing processes. Moreover, all the ingredients required to boost the interactive potential of images are ready at hand as intermediate results: 3D models, images of higher resolution, or documents with layers that separate primary graphical elements from secondary or structural elements, such as textual annotations, anchor points, connecting lines, scales, legends, etc. A direct connection to an underlying 3D model would enable viewers to select another point of view, but even static 2D images offer new possibilities for interactions: an unequal scaling of primary and secondary elements can guarantee the readability of textual annotations and authors / viewers should be able to change content displayed within the textual annotations. Recently the term *on-demand media* has become popular—the preference to control the media the way clients want; hence the emergence of personalized documents. The system may collect users' preferences and their browsing history over the time to format a unique edition of document for them.

It should be apparent from the above discussion that there are fundamental differences between the print and digital media. In particular, the dynamic nature of digital media restricts that the layout of the online documents cannot be predetermined for a particular content/user. The changes in information and users' intention make it necessary that the design problem is solved over the time at client workstation. Since this task cannot be carried out in real-time by hand for each individual user, an automated design process is required which can solve the layout problem dynamically.

This vision motivates us to propose a generic adaptive document framework that can produce dynamic design solutions while taking into account the necessary contextual requirements. Aiming towards the rich illustrative content we explore the embedding of interactive illustrations into the documents which are self-adaptable in the local as well as global context.

1.2. The Road to Adaptive Digital Media

An adaptive document framework would provide the designers with a unique strategy to manage the continuous changes of context in digital media and solve the design problem dynamically over the time. The new approach considers the document as a dynamic entity. Run-time changes in the content and user activities are reflected through the content/layout adaptation of individual items as well as global placement of elements in the document. This approach, at first, would reduce the burden in authoring documents. Designers would no longer have to create layouts for individual users and content type. Instead they can create a few examples and specify how the layout of the document should be adapted under various circumstances. At client's end, a dynamic layout engine would take the responsibility of formatting the document whenever the change in context occurs.

Adaptive document framework supports the human designers to describe a set of dynamic design solutions that can reflect the continuous changes in both the readers's intention and the content. To design such a framework, the designers need to anticipate potential changes in the context and analyze what actions should be taken in order to maintain effective communication of the information. Ishizaki [Ish03] points out that digital communication has three different characteristics when compared to print media (see figure 1.2):

- dynamic changes in information,
- dynamic user interaction, and
- temporal presentation.

Dynamic change in information brings out a new challenge in the digital documents. Online information systems, presently, allow the information to be quickly updated which requires that the presentation design is able to accommodate the updated information during the life-cyle of a document. Usually the designers achieve this goal by creating suitable design templates, thereby partitioning the document into its principal layout elements. However the templates can themselves be a problem. A template imposes a specific design, thus restricting the range of possible design expressions.

Second characteristic of digital media is interactivity. Information recipients may wish to personalize their documents according their interests. The personal selection of information and presentation style have to be determined separately for individual users. Since the reading pattern of users cannot be exactly determined in advance, designers have to

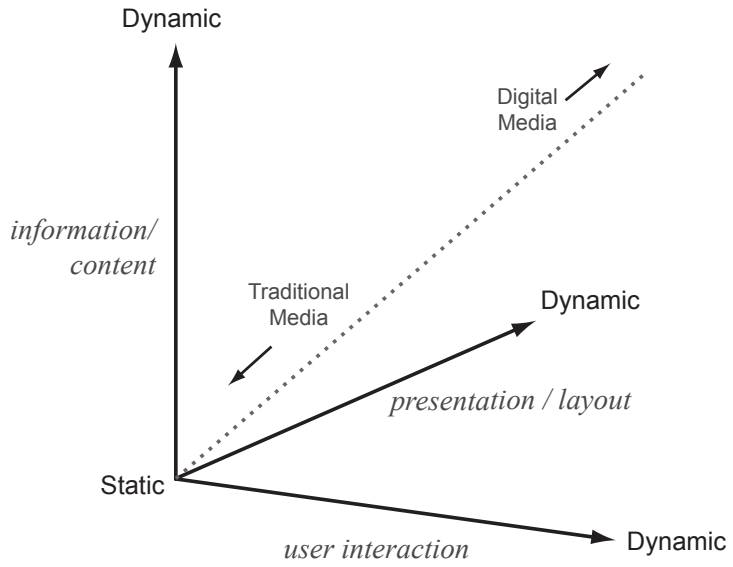


Figure 1.2.: Digital media differs from traditional media in three different respects [Ish03].

anticipate the users' behaviour in advance and give an explicit description of how to respond in various conditions. Furthermore, in order to aid the learning process it is very important that various type of media are seamlessly integrated into the document. However there has been limited research in context how the interactive media should be embedded within a complex document layout. Often the interactive visualizations are explored either within independent applications or inside applications which separate the visualization and the descriptive text in special panels. The spatial distance between them impedes the mental integration of co-referential or related information.

Third aspect in digital media is the capability of temporal presentation. Rather than relying on fixed forms of information, authors can embed multimedia-rich content such as animations or interactive visualizations, etc. to increase the learning efficiency. Since the continuous information updates and user interaction would require layout adaptation over the time, visual discontinuities in the presentation may occur, for example, content items swapping their positions or 'jumping' dynamic labels of an animated object. The effect of such temporal discontinuities should be reduced, otherwise it can be disturbing to the eyes and hinder the learning process. When the layout changes are absolutely necessary, such transitions should be animated.

These characteristics of digital media demand new techniques to extend the static nature of traditional media into a more flexible and responsive one. Thus an adaptive document framework must address the issues of dynamic changes of context and temporal presentation

in order to generate dynamic design solutions. The framework would require a close collaboration between authors and designers in preparation of content and a set of suitable visual designs that can be manipulated as per requirement by the system.

1.3. Contributions

A novel idea is conceived to develop a generic adaptive document framework. In this framework, the design of digital documents is adaptable to the changes in content and viewing. The framework not only provides unique solutions for overall adaptation of document layout, it also addresses more specific problems, such as layout adaptation of the individual illustrations within the document.

Figure 1.3 depicts the adaptive document framework. It consists of three major components which basically reflect the three fundamental characteristics of digital media (cf. figure 1.2) as discussed in the previous section. *Content authoring* tool allows the authors to prepare different versions of the content and define the structural representation. *Layout engine* is responsible for arranging the content items in the document. It analyzes the continuous contextual requirements to create new design solutions from the given templates. As a third component, the *document browser* renders the information and provides a platform for the clients to browse the document and interact with it.

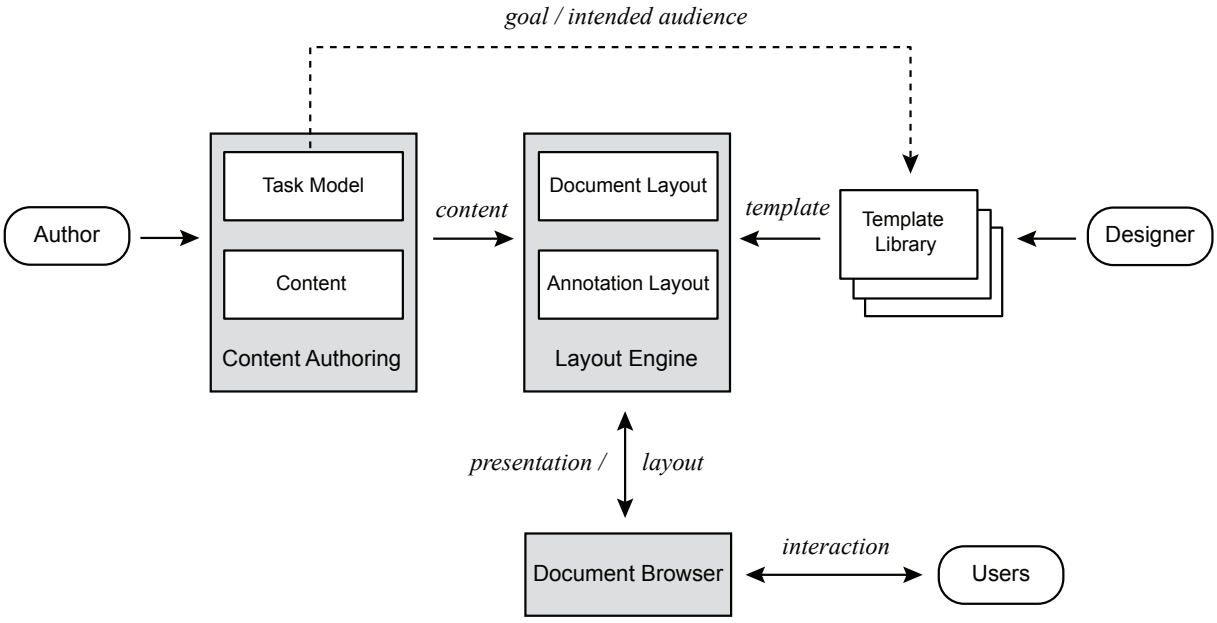


Figure 1.3.: Adaptive document layout framework.

Authoring: The strength of an adaptive document layout framework lies in the content and style representation. Therefore, in the presented work, the content and style representation are comprehensive which take into account the adaptation aspects. For content adaptation, a scheme is defined for authors who prepare the content at multiple granularity levels that can be used to create different versions of documents. Document content, such as text, static illustrations, animations, and interactive 3D illustrations, is represented in XML tree structures which enables hierarchical organization of data.

An template authoring tool is provided to the designers to create designs patterns (known as templates). The templates encode graphical constraints which are resolved by the layout engine at run-time, and are stored in the *template library*. As one of the key differences from the past research in this framework, the templates have the capability of being much flexible. They are no longer static template, hence, they can be adapted to various screen size and varying content.

Using a novel task-driven approach, the authors can specify different views of underlying content and indicate the relevance of particular objects in illustrations with respect to the task at hand. These tasks are stored in a *task model* used by layout engine to filter content and choose appropriate template to lay out the information.

Layout Engine: A dynamic document layout adapts and formats the document content automatically. We introduce a unique combination of constraint-based and force-directed layout methods [AHFS08] to generate dynamic design solutions from the given templates. Force-directed algorithms provide an effective means to achieve an optimal placement of content even if the layout templates are partially designed or absent. To achieve this, the content and layout requirements are modeled through a set of physical forces to change the position and size of graphical objects in the page. The forces work hand-in-hand with designer-specified constraints to make adequate changes to the layout. While the past research in the document layout has focused on grid-based design, the force-directed approach is able to yield complex non-grid layouts suited to rich illustrated material.

In particular, the layout engine incorporates an *annotation layout* component to facilitate annotated illustrations within the documents. In this work, the annotation layout problem is dealt by developing heuristics that take into account various

aesthetics and functional attributes from the hand-made illustrations and interactive media [HGAS05, GAHS05b]. The proposed heuristics deal with the placement of annotations inside as well as outside the graphical objects. Furthermore, they are equally suitable to annotate 2D illustrations as well as 3D interactive visualizations [AHS05, GAHS05a, FLH+06] dynamically.

1.4. Thesis Organization

Chapter 2 - Related Work

Chapter 2 gives an overview of the state of the art work in the field of document layout. The chapter starts with the definition of the document, compares digital documents with paper media, and then it proceeds with the discussion of document layout in digital media. The chapter is divided in three major sections i.e., content representation, automated document layout and document browsing. The core of the chapter is the automated layout section which discusses layout techniques for documents as well as for annotation placement in graphics. This section also includes the discussion about pagination and line-breaking.

Chapter 3 - Conceptual Framework

This chapter introduces a conceptual design for an adaptive document layout framework and describes various components of the system.

Chapter 4 - Authoring Documents

Chapter 4 is about authoring adaptive documents. It is comprised of three major parts. *Content authoring* part presents the content representation in the proposed system. The second part is about *layout authoring* which provides a detailed description of how the adaptive layout can be authored. A template authoring tool is provided for the designers to create design templates that can be continuously adapted to the layout requirements of dynamic documents. In the the third part of the chapter we discuss the *task models*

within the scope of dynamic documents. Our novel approach employs the task models to store guidelines for selection of content and layout in order to achieve particular tasks.

Chapter 5 - The Layout Engine I: Adaptive Document Layout

In chapter 5, we describe the way of determining the adaptive document layout. The content and template selection process is described along with a discussion of flowing the content into pages. In addition to the traditional constraint-based layout approach, a novel approach is introduced that integrates the forced-directed techniques with constraints, to perform the layout adaptation.

Chapter 6 - The Layout Engine II: Annotation Layout

Chapter 6 presents a novel approach for automated placement of textual annotations in the interactive 3D illustrations. Based on a corpus study of hand-made anatomical and instructional illustrations, several annotation layout styles were classified. A large part of the chapter is devoted to the heuristic procedures that implement these annotation layout styles.

Chapter 7 - Conclusion

The last chapter concludes the thesis and discusses the potential areas of future work.

Chapter 2

Basic Concepts and Related Work

Computer technology has revolutionized the way information is created and shared among individuals. Information in the digital network can be updated regularly. The information recipients are a wide variety of users who are connected to the web via different devices and have their own requirements. Traditional methods of document creation and delivery are inadequate in digital environment because such approaches are directed towards producing static documents. The dynamic nature of digital media requires methods to automatically create documents tailor made to user needs. The current tools of authoring multimedia documents are either non-automated or provide limited support in specifying flexible designs. Thus, there is a strong need to develop new methods for producing content and layout automatically for multimedia documents.

This chapter reviews the state of the art research and development related to the field of automated document layout. The chapter starts with the definition of document and evolution of document production in traditional and digital media (see section 2.1). Different content representations of documents are discussed in section 2.2. Content formatting and automated layout approaches are presented in section 2.3. Section 2.4 covers the topics related to document browsing and reviews the interaction and personalization schemes in digital documents. Section 2.5 concludes the chapter.

2.1. Background

The American Heritage Dictionary defines a document as "*a written or printed paper that bears the original, official or legal form of something and can be used to furnish decisive evidence or information.*" The dictionaries commonly view documents as body of information on paper. The term "document" may refer to a particular physical rendering of a certain body of information (e.g., a particular print of the Bible). In contrast, a document can also denote a body of information without suggesting how this information is physically instantiated (e.g., the Bible, not a particular copy of it). Thus "document" bears multiple meanings—*container* or *contents* [Nun79]. According to this, a particular physical rendering of information (i.e., container) can be regarded as document. Similarly, a certain body of information alone (i.e., contents) can also be a document. Both are notions of a document, albeit two different ones. The former is a static representation of information, and the latter is able to evolve over time and capable of rendering multiple instantiations from the same pool of content. It is important to note that, in the end, all information must be delivered in tangible form. Therefore, documents always exist either in physical form or rendered electronically [Buc97, Buc98]. Different representations of a document may describe same facts using different characteristics [Buc91]. They are considered as different renditions of the same document.

2.1.1. A Brief History of Document Production

The history of evolution of documents is spanned over a long period of time. Cave man made use of leaves, animal skins, stone tablets and marking tools like sharpened-stones and knifes to make drawings, signs and words. About 800 BC, the Greek alphabet was developed and first examples of manuscripts, which used only text, were written. First reliable ink was developed around 400 AD and first true paper was invented in China in the 2nd century AD. Europe started paper production much later, in the 12th century. Early documents were *scrolls*, rolled up pages glued together. Scrolls could be rolled and unrolled from side to side to expose one page at a time, thus they allowed only sequential reading. Eventually scroll documents were replaced by *codex*, a modern format of book with separate pages bound together from one side. The advantage of codex was that it allowed quick flipping through pages and random-page access. Till the 15th century there was no proper mechanism to produce documents in mass quantities. The documents were

either copied by hand or inked wood-blocks were used to make impressions on the printing surface.

Modern printing began when Gutenberg invented *movable typesetting* printing press in 1440 to enable mass production of documents. In Gutenberg's Printing Press, ink was applied to the movable type block letters which were pressed on the paper to make a print. After about 300 years, in 1796 *lithographic* printing process was developed which used flat plates and chemical principles to reproduce images for printing. The 19th century saw the invention of *typewriter* and *carbon paper* which were widely used for writing and copying manuscripts. In copying technology, Xerox copy machine, released in 1959, was a huge success for duplicating existing documents.

As computer technology advanced, desktop publishing began in the 1980s. In desktop publishing, authors began to use computer hardware and software to create documents. Markup languages such as TEX, which focused on text-formatting, allowed the authors to typeset documents and especially write complex mathematical formulas. Later on, WYSIWYG (What You See Is What You Get) word processors like MS Word simulated the page layout on the screen. Other commercial desktop publishing software such as QuarkXPress, Adobe InDesign and Microsoft Publisher allowed the authors to create even complex page layouts for text and graphics. But the results are static documents that must be viewed at fixed size. The arrival of World Wide Web posed new challenges to the document authors. Web content may be dynamic and web users may have different needs and connected to different display devices. The web requires the automatic adaptation of documents according to the viewing context. HTML and CSS provide a way to separate content from presentation and specifying layout parameters, but the approach is limited. Though web media have become very dynamic these days, web browsers produce layouts that are inflexible to the changes in context. The research is still on to make web media effective for a wide range of viewing scenarios.

2.1.2. Paper vs Digital Documents

In 1990 when the computers and technology assumed important roles in the workplace, executives at Xerox PARC envisioned that eventually technologies were going to replace the paper, leading to the *Paperless* office. Surprisingly the opposite has happened. The technology has facilitated even the greater use of paper. Now it is estimated that an average office worker consumes 100 pounds of paper per year. Some people view the use

of traditional paper media as the sticking to the old habits and our stubborn resistance to efficiencies offered by computerization. But Sellen and Harper [Sel03] do not agree. They argue that the technology was not able to replace the paper because when it comes to certain kinds of cognitive tasks paper is better suited than computers. According to Sellen and Harper, paper offers a unique set of *affordances* that allow specific kinds of uses. Due to the following affordance features paper-based reading still remains a popular choice:

- *Flexible Navigation:* Paper is tangible meaning one can pick up the document, flip through pages and quickly get a rough idea about the written material. Sellen and Harper observed that except when reading a novel most of the people do not follow a strict sequential order of reading a document.

- *Spatially Layout:* Paper documents can be spread out, arranged or folded in the way we want. In contrast screens provide a restricted field of view for multipage view.

- *Annotation:* Paper can be easily annotated during reading and key information can be highlighted.

Paper is treated as a physical evidence to a fact [Buc97] and used as backup. Despite the fact that computers allow the documents to be efficiently stored, searched, linked, shared and accessed remotely, case studies have shown that the benefits of paper exceed those of digital documents [OA97]. Hansen and Haas found out, on several factors that readers of on-screen documents may have difficulty in getting the *sense of text* when the text is long and complex [HH88].

It is interesting to note that screen size, resolution, contrast or viewing angle are not important factors separating paper and computer. Instead it is due to the lack of affordances in digital documents that people will continue to use paper for reading purposes. Sellen and Harper discuss that there are, however, other reasons why one might want to avoid or reduce the use of paper as:

- *Symbolic:* Papers are old-fashioned.

- *Cost:* High printing and maintenance costs.

- *Interactional:* Papers cannot be accessed remotely, rather only locally. Paper documents are in physical form, thus they require physical delivery which makes

mass-distribution a time-consuming and expensive task. Moreover, the document replication is costly and the fact that the paper documents are static in nature.

These limitations of traditional media are effectively addressed by digital media that possess following modalities [Nah]:

- *Media Integration:* Integrating text, figures, sound, animations and 3D visualizations.
- *Constant Originality/Reusability:* Original content can be modified to create new content. Reproduction is cheap.
- *Boundaryless Documents:* Content in digital media can be indefinitely associated using hyperlinks, so the content is in continuum.
- *Collaboration:* Remote sharing, collaborative authoring of documents.
- *Interaction:* Dynamic content and presentation adaptation based on user queries.

In order to further boost the potential of digital documents some researchers have tried to mimic paper-based techniques in the digital documents. PARC PADS is based on such idea. Pads differ from desktop computers in a way that Pads are small-sized computer which are intended be "scrap-machines" that can be grabbed and used anywhere just like POST-IT notes [Wei91]. In a complex PAPERWINDOWS setup, the GUI windows are projected on the physical paper [HVA+05]. By tracking hand gestures the system can simulate the affordances of paper in digital environment. In [DAK+06] a different setup is used to eliminate the artificiality of the digital environment. This is done by allowing the users to manipulate the digital documents directly using the haptic feedback devices. However a question remains open whether such techniques can be developed to an extent that they can *really* boost the learning experience. In a case study TERRENGHI et.al. compare physical and digital manipulation on a tabletop to evaluate interaction on objects affordness [TKSI07]. They conclude that physical metaphors of input may improve the user interaction with the virtual environment, but a deeper understanding of affordances and interaction is required in the digital realm. The work on tangible user interfaces is an ongoing research. Currently such interface techniques are domain specific and some serious applications aside they are often used in fun projects. The problem is tangible interfaces require a special setup (often requiring large displays) which means majority of the World Wide Web readers won't have this luxury for browsing documents.

The fact that a wide variety of display devices with different capabilities can now access web poses a big challenges to the authors who wish to support access for all devices. The difficulty lies not only in adjusting the display layout but also in transforming the content according to device capabilities. Therefore, automatic adaptation of digital documents requires the authors to deal with three major issues: *content authoring/management*, *presentation layout authoring/adaptation* and *document browsing and interaction*. We are going to discuss these topics individually in the coming sections.

2.2. Content Authoring

Content management is an important issue for dynamic systems where the information is updated according to the varying context. Different readers might ask for different versions of content and adapting layout for different screen sizes might require alternate content. In order to handle these issues the document specification should represent the content in a well-organized manner.

First step towards adaptive documents is separating the document's content from presentation style. World-Wide-Web Consortium (W3C) style sheets support this separation. Markup languages such as (HyperText Markup Language) HTML or (Extensible Markup Language) XML [Rec06a] are used to define the structure and semantics of content using tags. On the other hand the layout or the style of the presentation is defined separately in the style sheets using languages such as Cascading Style Sheets (CSS) [Rec98] and Extensible Stylesheet Language (XSL) [Rec06b]. Authors can design a series of style sheets targeted for different media types. For example, a style sheet for "`screen`" that applies to desktop displays, a style sheet for "`handheld`"' intended for mobile devices, and a style sheet for "`print`"[1]. The mapping of content to style is done by matching the markup tags (e.g., `title`, `body`, `img` etc.) associated to the input material with the corresponding tags in the style sheet. By decoupling the style of a document from its content, XML repositories can generate different views of the document for different readers, all from a single source document. This approach can improve maintenance, portability, consistency and customization of documents because content and layout can be dealt *relatively* independently.

1 For complete list of media types, see `http://www.w3.org/TR/REC-CSS2/media.html`

W3C standard XSL Transformations (XSLT) allows to extract data from source documents and allows the conversion of XML documents to other formats like XML and HTML [Rec07], thus offering a possibility to reuse existing content in different formats. Furthermore, W3C provides Document Object Model (DOM) which is a platform- and language-independent interface for representing HTML / XML and related documents. In DOM, a document is viewed as a tree-structure with elements, attributes and text as nodes. Application programs and scripts can use DOM interface to access and modify the content and style sheets dynamically. Thus, DOM is very suitable for interactive systems where the information is inspected repeatedly or out of sequence.

As for authoring content for online documents, XML is a popular choice because of the content reusability. There are several tools in the market to make XML-based authoring easy. In XML editors based on WYSIWYM (What You See Is What You Mean) paradigm, authors create content in a structured way focusing on semantics rather than the actual formatting. An example of WYSIWYM editor is Butterfly XML. On the other hand in WYSIWYG (What You See Is What You Get) paradigm the XML content displayed during editing process corresponds to the final formatted output. A WYSIWYG XML editor such as Vex (Visual Editor XML) applies the style sheet (XSL) while author is editing XML content[2]. So instead of seeing XML tags or attributes the author sees formatted output as defined by the style sheet.

Though the style sheets offer a possibility to define the presentation style on different media, they are not the ultimate solution. In addition to authoring layouts, adaptive documents have to also support appropriate content authoring and content selection. Characteristics of display device might require choosing alternate content when certain content does not fit well on a different screen. For example a document designed to be viewed on desktop screen might contain images which may require a considerable amount of scrolling when displayed on screen with different size or aspect ratio. In this case we might need to consider different crops or size of images. XML/HTML and style sheets provide limited functionality when it comes to designing adaptive content/layout for a variety of devices. Instead most research work and commercial products focus on *content reformatting* in order to make content more readable on the constrained display device [BS97, Sof, Bit], for instance resizing fonts and rescaling or deleting images so that the content fits well on screen. When reformatting content, detection of the roles of images is very important

2 A comprehensive list of WYSIWYM- and WYSIWYG-based XML editors is available at: `http://www.xml-dev.com/xml/editors.html`

in order to process documents correctly. One such method of automatic classification of images from web pages is given in [MHN06]. A disadvantage of content-reformatting methods is that they tend to destroy the original page layout and readers may have to hunt for content.

It is apparent from the preceding discussion that adapting the layout for wide range of displays also requires adaptation in content. Furthermore, the readers on World Wide Web may come from different backgrounds, therefore, different versions of content may suit them. Mohan et.al. proposed a method to adapt multimedia internet content [MSL99]. They use InfoPyramid to represent content encoded at multiple resolutions and modalities (see figure 2.1). When the system receives a request, it automatically selects from InfoPyramid, the content with the resolution and modality to optimally match the capabilities of client devices. Similarly Li proposed multi-representation of content [Li02]. Authors create multiple versions of every type of content that can appear in the document, such as text, images, side-bars etc. The multi-representation content is stored in a tree structure that has two types of nodes: *content nodes* which store actual content, and *structural nodes* which combine multiple versions of content (see example in figure 2.2). Using this representation, authors can create content at multiple levels of details and different geometric properties. During the layout process, document tree is analyzed to select alternate content automatically. The selected content is then formatted to match the display device [JLS03a].

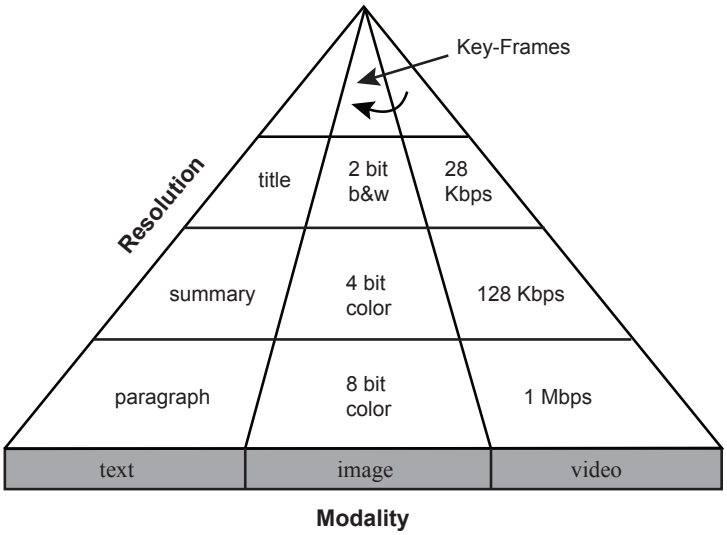


Figure 2.1.: Using InfoPyramid to encode content at multiple resolutions and modalities [MSL99].

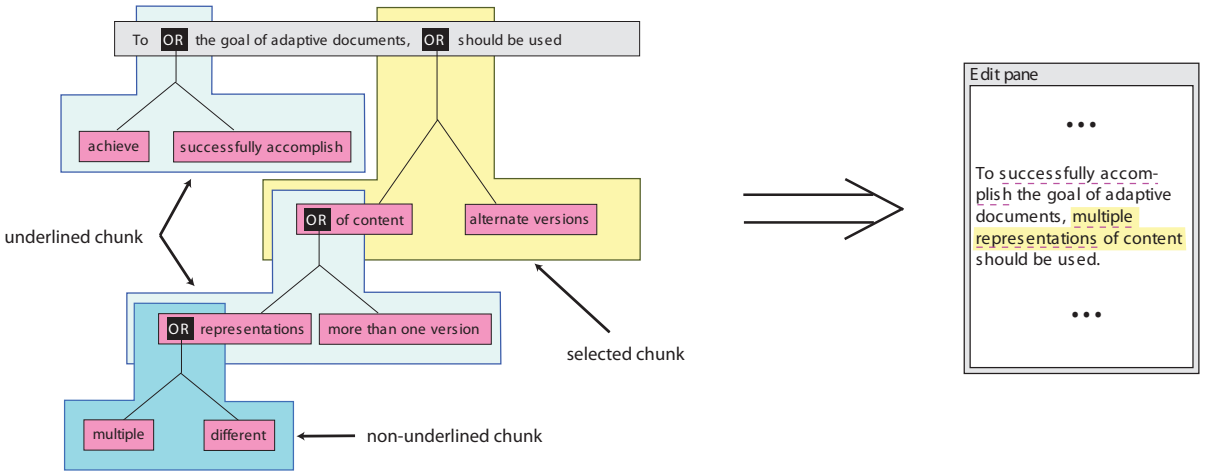


Figure 2.2.: Multi-representation content tree with a corresponding text view [Li02].

As promising as multi-representation technique is it purely relies on manual authoring of alternate content for different viewing scenarios. Though flexible this approach could be very time-consuming. There has been a lot of research in the field of information retrieval with a focus on *content extraction* to automate authoring process. Unlike authoring multiple versions of the same document, automatic content extraction techniques attempt to extract useful and relevant content from existing web documents. One such technique works on DOM tree of HTML documents to filter content by taking user preferences into account [GKG+05]. Other researchers have used visual representation to automatically extract the content structure [CYWM03a, CYWM03b] of web documents. Vision-based content extraction improves the traditional DOM based content extraction methods by combining the DOM tree and visual cues that separate content items.

Another area of concern in authoring online media is *document localization*—adapting documents to different languages and regional differences. To make the documents understandable to a large audience who is spread geographically and speak different languages, the content must be translated to other languages. Typically localization is achieved by authoring an XML file containing the language resources under different subtrees and using XPath language to locate the desired content within XML document. However manual translation of documents adds burden on authors. Also the traditional approach of creating multilingual XML documents under control of Document Type Definition (DTD) puts certain limitations on expressing semantics between the subtrees of document structure. Furthermore, styling languages such as CSS and XSL are not fully equipped to handle linguistic differences. In contrast MDA (Multilingual Document

Authoring) system [BDL00, DLR00] features an authoring process which is monolingual but can create multilingual texts. Authors build document representation in their own language and the system builds up translated versions from content representation in as many language as it supports. The proposal is that an XML document should be *surface-free*, meaning there should be no text between `<tag> </tag>` items. Rather the nodes in the document content represent meaningful types and labels. MDA requires the authors to create an initial development of the grammatical resources and conversion scheme for all supported languages. Translations are possible for only those parts of document which conform to the formal grammar specification.

Another vast area of research focuses on extraction of content from scanned paper documents. An example is WISDOM++ system [AEM01] which employs several techniques—OCR, document layout analysis and text formatting etc. — to transform scanned documents to HTML/XML format. Such system often use machine learning techniques to improve the layout analysis [MEA+03] which is a crucial step in the context extraction process.

2.3. Automated Layout

A graphical layout problem in general deals with determination of positions and sizes of a set of graphical objects in the presentation. The term presentation refers to any material that is intended to be viewed or manipulated by the user. The examples are graphical user interfaces, World Wide Web documents, even conventional newspapers and magazines. Layout is a crucial part of every presentation as it plays a major role in how well the information is communicated to users who interact with it. A well designed layout would boost the reading-speed and information within the document would be better understood and retained in the memory.

In this section we review state of the art work in the field of automated layout. Three subsections reflect three levels of topics. Top-level is *document layout* which is responsible for determg the size and position of graphical elements in the page. Underneath is *content layout* which considers flowing content into regions—splitting content into pages and breaking paragraphs into lines. Bottom-most layer is *annotation layout* which deals with the labeling problem for individual figures in the page.

2.3.1. Document Layout

Creating a presentation layout requires human skill to design. By design, we mean the arrangement of visual objects displayed in the presentation subject to some given restrictions, e.g., limited allocated space, non-overlapping content items, as well as further aesthetic and functional criteria such as symmetry, visual balance, or readability. Therefore, even today this challenging problem is solved by expert human designers who spend extensive time in accomplishing complex document layouts.

To save effort designers often create a set of design templates which can be modified to produce different versions of layouts over the time. The *template-based* approach is quite common in print media such as newspapers, magazines and books because the layout parameters (e.g., page size, number of columns, type size) are fixed and the content to be laid out are known at the time of printing. While static layout are unavoidable in printed media, the layout for digital media need be more dynamic. Constantly updated content and viewing requirements demand that the digital documents should have the capability to adapt themselves according to the continuous changes in context. In this section we discuss state of the art approaches which deal with the problem of computing layout for digital documents.

2.3.1.1. Quality Considerations

Automating the design for graphical presentations in general is a hard task. This difficulty arises because of two reasons, as pointed by Mackinlay [Mac88]:

- We must encode design criteria in a way machine can understand it *(expressiveness criteria)*.
- Presenting information on output medium in perceptually effective way *(effectiveness criteria)*.

According to Mackinlay the effectiveness criteria of a graphical presentation can be evaluated from various factors, e.g., if the design can be interpreted accurately or quickly, its visual impact, and its computational complexity. Mackinlay developed a presentation tool APT to automatically generate effective graphical presentations [Mac86] by incorporating Bertin's vocabulary [Ber83] and the visual perception effectiveness criteria [CM84]. Although Mackinlay's criteria was aimed towards automating design of graphical presen-

tations based on relation data (e.g., charts, scatterplots and graphs), the criteria remains the same for adaptive layout for documents too. This means, the designers must be able to encode layout requirements that can be computationally resolved to generate new design solutions. Also the new layout designs must be meaningful and provide a means of smooth dialogue between the reader and the presentation.

Beach [Bea85] pointed out that the general layout problem for determining whether an arbitrary set of non-overlapping rectangles can be arranged into a minimum rectangular space is NP-complete. Thus, there exist no efficient algorithm to solve it. Even though the the layout based on the space-minimization criteria might be very space-efficient, it can lead to a poor design which is difficult to use and understand. The layout problem becomes even more complex when additional aesthetic criteria is added to it. The challenge is to develop methods that generate effective layout according to the evaluation criteria in a cost-effective manner, i.e., reducing the number of possibilities that are evaluated. Online media is even more challenging because of the varying content and context that require flexibility in design. Since digital documents are supposed to be interactive they also impose constraint on time, hence simple yet efficient solutions are needed.

Judgment of design aesthetics (on subjective basis) is a task that comes naturally to humans. Most of us are good at deciding how good a page layout is after looking at it. Yet, is is much more difficult to describe a criteria which will give rise to aesthetic design. However, in order to incorporate aesthetic aspects in online document, it is important to have some general guidelines towards pleasing page layout. Some principles towards aesthetics of layouts: organization, simplicity, contrast, balance, and harmony or unity are discussed in [Reh86]. Harrington defined practical aesthetic measures which can be applied be document layout [HNJ+04, HNJ+06]. Their approach combines following heuristic rules that can affect the aesthetics:

Alignment: Content objects should be displayed in an aligned pattern.

Regularity: Aligned positions should be spaced in a regular fashion. For example, row and columns of a table have relatively same height and width.

Uniform Separation: Objects are uniformly spaced.

Balance: Visual balance of the page layout is determined with respect to *visual weight* and *visual center* of the page. Visual weight of the an object is calculated from

object's area and its opacity. Visual center is located near to the geometric center of the page.

White-Space Fraction: white space (including margins) should be total about half of the total page area.

White-Space Free-Flow: White-space should always be connected to the margins. White-space trapped by surrounding objects should be avoided.

Proportion: Proportions of all major content elements and theirs groups are compared with the idea *golden ratio*.

Uniformity: Content objects should be distributed uniformly over the page.

Page Security: Small objects should not be positioned at or near the edge of the page.

Rules are defined for individual pages and assigned relative importance. Scores from each page are then combined to calculate the overall document measure. In oder to encode such design rules in the layout, graphic designers can create layout templates. Layout templates store a set of graphic rules (constraints) that must be fulfilled to generate a valid layout.

2.3.1.2. Constraints as Design Criteria

In any layout task, there are restrictions that specify how the information could be laid, and there are measures to judge the usability or aesthetics of the layout. For instance, a common layout restriction can specify that the title should be placed above the body text, another example is that the presentation layout should not exceed the area of viewing device. Constraints provide an intuitive way to encode restrictions on layout. A constraint is simply a condition that the layout must satisfy. A set of constraints can be used to specify a page layout, including layout dimensions, figure placement and columns width etc. Constraint can be tagged as hard or soft. Hard constraints must be necessarily met, while soft constraints describe preferences of varying degrees of importance. Given a set of constraints, the goal of constraint-based optimization is to compute a layout that satisfies all constraints.

Constraint may be forced by the author as well as the by the viewer. A designer might impose a particular appearance for a document. On the other side, clients may want to personalize the document in their own way and have varying device capabilities. Hence, the final look of the document is a compromise between the author and viewer.

Definition: A graphical constraint is a logical relation among several variables, each taking a value from a particular domain. A constraint restricts the possible values that variables can take. Constraint can be employed to declare screen and page layout, specifying relationships among the layout elements, figure placement, column widths, spacing and margins as well font sizes. Constraints allow us to specify *what* relationships must hold without, rather than *how* these relationships would be enforced by the system. Another advantage is that constraints may specify partial information of the layout, i.e. the constraint need not to specify the unique values of the variables, ($x >= 5$ does not give the exact value of variable x, so x can be equal to 5,6,7, etc.).

Constraint Types: In a constraint-based layout system, constraints can be categorized as either *abstract* or *spatial*. Abstract constraints encode the high-level semantic relationships between various items to be placed in the layout. Some examples are `FIG1 is IMPORTANT` and `TEXT1 REFERENCES FIG1`. Abstract constraints are specified by the author while preparing the content. In contrast, spatial constraints express geometric relationships among the components in layout. This is usually done by declaring position and size restrictions on the graphical objects. Furthermore, spatial constraints can be either *absolute* ones, with constant geometric parameters, or they can be *relative* thus relating the geometric parameter of object to another. Examples of absolute spatial constraints are `TITLE XPOS = 10` and `WIDTH-OF-FIG1` $>$ `50`. Some relative spatial constraints are `CAPTION1 BELOW FIG1` and `WIDTH-OF-FIG1 = COLUMN-WIDTH`.

Since abstract constraints do not specify the position and size of objects they must be translated into spatial constraints before the layout process can begin. This is usually done by mapping each abstract constraint into a possible geometric parameter. For example for an abstract constraint, `FIG1 is IMPORTANT`, `FIG1` can be assigned dimensions large than default figure size, moreover FIG1 can be placed on the main page. Similarly for an abstract constraint `TEXT1 REFERENCES FIG1`, `TEXT1` and `FIG1` can be placed spatially close to each other.

Constraint Representation: In a graphic design containing several layout constraints, there should be a language to express constraints on visual elements. Constraints can be specified using relational grammar rules such as `Below (CAPTION1, FIG1)` [WW94]. Though the grammars are a powerful way to express layout constraints it may be difficult to specify all possible relationships between the presentation elements and complex grammar solvers might be required. Another more efficient way to express constraint rules is to use

Boolean predicates which narrows down the search space. This approach is demonstrated in Graf's LayLab [Gra92].

Next question is how to obtain constraints. Most systems such as Weitzman's system [WW93] and Graf's LayLab use graphical interfaces to let the users allow specify constraints for multimedia documents. On other hand there has also been some research in automated extraction of constraints from example layouts. But most of these techniques have been tried only in building graphical user interfaces [MMK93, MGD+90, MMT+92], graph layouts [Mas94] and to generate diagrams [ZM01].

2.3.1.3. Constraint-Based Layout

In constraint-based approach document layout is modeled as constraint satisfaction problem where a set of constraints specify the design criteria. Layout styles/templates encode such constraints, and therefore, contain variables, a value domain for each variable and relationships among variables that must hold in the final layout. Style sheets are an example of document layout templates. Commercial softwares such as Microsoft Word, PowerPoint, Adobe InDesign, Quark Express and LaTeX provide a set of predesigned templates to choose from and the ability to create ones, that can be applied to format the content. In [JLS+03b] authors can create a set of templates for different display sizes which can be automatically adapted to a range of viewing conditions.

Given a set of constraints in the template, constraint satisfaction process attempts to resolve the constraints and assigns each variable a particular value from its value domain [Mac92]. For document layouts, this means determining the values of size and position variables of each element in the layout. There are a whole bunch of constraint solving techniques, only a few of them are discussed here within the scope of document layout.

Grid-Based Layout: In graphic design, typographic grids are frequently used as an ordering system to efficiently organize text and illustrations [MB96]. Newspapers and magazines are examples of grid-based layout. When the layout is performed in a grid-based fashion, the visual elements in the layout are sized in proportion to the size and aspect ratio of grid cells and aligned with the grid lines. In this way a grid adds further constraints for the positions and sizes of components, thus narrowing down design search space for layout arrangement and making the layout problem computationally tractable.

In GRIDS [Fei88], the system automatically generates an underlaying grid based on the display size and the type of content to be displayed. Then the grid is used to format the content within the grid cells. In another grid-based system [JLS+03b], typographic is drawn by hand and is used as a guide for the author to input spatial constraints. The constraints are then solved using constraint propagation methods.

Linear Programming and Constraint Propagation: A layout problem is conveniently described as a collection of linear equality and inequality constraints. For example an inequality constraint `fig2.right>fig1.left` indicates the requirement that `fig2` is placed to the right of `fig1`. Similarly, an expression `fig2.width-fig1.width=0` describes an equality constraints.

A system of linear equality and inequality constraints can be regarded as a linear programming problem. Consider a set of n real-valued non-negative variables variables $x_1, ..., x_n$. The set of m constraints can be written in the following form:

$a_1x_1 + ... + a_nx_n = b$
$a_1x_1 + ... + a_nx_n < b$, or
$a_1x_1 + ... + a_nx_n > b$

The goal of a linear programming optimization is to maximize (or minimize) the *objective function*, subject to linear equality and linear inequality constraints. The objective function can be written as

$c + d_1x_1 + ... + d_nx_n$

Solving a system of constraints has been an active area in interactive graphical system beginning with Sketchpad [Sut63]. One of the most popular algorithm for finding a solution of the linear programming problem is *simplex algorithm* [Nas00] that minimizes the value of objective function. In practice, many systems use one-way constraints or local propagations methods to solve multi-way constraints [SMFBB93]. Though local propagation methods are unable to solve simultaneous equations, they are very general and efficient. In general purpose constraint solvers, each constraint is specified as a linear equation or inequality [BBS01]. Therefore, layout algorithms need to translate high-level graphical constraints into linear constraints. Examples of systems which do this transformation into low-level constraints are [WW94, Gra92].

Force-Directed Techniques: Constraints can also be implemented using force-directed techniques. In a force-directed *spring-mass* setting, the layout elements are modeled

as nodes (mass) and layout constraints are modeled as edges (spring) between nodes. Fruchterman and Reingold [FR91] proposed a spring-embedding approach for drawing graphs [DETT99]. Their method works on a simple principle:

- Vertices connected by an edge should be drawn near to each other,
- Vertices should not be drawn too close to each other.

These rules are modeled by repulsive and attractive forces. An attractive force enables the vertices to attract each other if they are edge-connected. Similarly other constraints such as non-overlapping or a desired separation between elements can be implemented by introducing repulsive forces between nodes. Given an initial setting, the system applies physical simulation model on the spring-mass graph.

Let us assume a graph G consisting of vertices V as the nodes of G and a set of edges V modeling relationships among V. In a spring-mass model, vertices that are far away are attracted to one another due to spring forces, vertices that are too close repel one another. According to Hooke's law, the force exerted by spring force F is calculated as:

$$F = -k * (currentLength - naturalLength)$$

k is the stiffness of spring, and $currentLength - naturalLength$ is the distance that the spring has been stretched or compressed away from its natural length. The negative sign in the equation indicates that the spring force acts in opposition direction of displacement. $naturalLength$ of spring depends on the use of spring, shape of object and the minimum vertex separation. Thus, if the spring is stretched from its natural length, the nature of force between nodes is attractive, and when spring is compressed, it is repulsive force. In this way, spring forces ensure that the edge-connected nodes are close apart on the page but do not overlap one another.

Forces are applied to the vertices, pulling them together and pushing them apart in an attempt to converge to an energy-minimizing state (see Algorithm 1) [FR91]. Force-directed algorithms tend to produce aesthetically pleasing layouts and exhibit symmetries. These methods have been frequently used in graph drawing [DETT99]. TEX uses a variant of spring-mass model to compute spacing between words in line-breaking algorithm, but to our knowledge until now no one has tried to use spring-mass model to compute document layouts. An extension to forced directed graph drawing was introduced for automatic placement and sizing of non-overlapping windows in a window-oriented graphical user

interfaces [LES95]. Rectangular windows are modeled as nodes which are placed on the screen area. In post-processing, the forced-directed makes changes to the positions of nodes, and pressure-directed phase modifies the window sizes. However their approach is very basic and the final output is similar to graph layouts with non-uniform vertices.

Algorithm 1 Force-Directed Layout

```
InitializeLayout()
for i = 1 to iterations do
  for each layout element node p do
    F_net ← (0, 0)
    for each other element q do
      F_net ← F_net + f_repulsion(p, q)
    end for
    for each other element node r spring-connected to p do
      F_net ← F_net + f_attraction(p, r)
    end for
    move element p in the direction of F_net
  end for
end for
```

Search Techniques: Another way to solve layout problem is to consider as a search problem. The goal is to search an optimal layout over a space of possible layouts. An evaluation function *scoreLayout*() is defined to assess the quality of layout on the basis of constraints at each step. There are a variety of techniques to solve the search problem [MF04]. *Random Search* considers making random choices at each step and checking if it leads to a better solution. A *Brute Force* search technique systematically evaluates all possible combinations to find a solution. In a *Gradient Descent* method, which is local search technique, that considers all the alternative placements for each element and manipulates the element if it improves the layout (see Algorithm 2) [CMS95].

Another well-known search technique is *Simulated Annealing* [KGJV83]. A randomized search algorithm that allows making poor moves at certain stages to avoid sticking into local minima. A parameter T temperature controls the amount of space that can be explored at any time. Initially T is high that allows to make big steps towards improvement, and as it gradually decreases according to annealing schedule, small steps are taken in the gradient direction (see Algorithm 3) [CMS95].

The difference between gradient descent and simulated annealing is that the latter accepts moves in directions other than that of gradient. This often results in escaping local minima.

Algorithm 2 Gradient Descent

```
InitializeLayout()
E ← ScoreLayout()
repeat
  for each layout element do
    move an element to a new position
    newE ← ScoreLayout()
    if new layout is better (newE < E) then
      E ← newE
    end if
  end for
until no further improvement possible
```

Algorithm 3 Simulated Annealing

```
InitializeLayout()
E ← ScoreLayout()
repeat
  for each layout element do
    move an element to a new position
    newE ← ScoreLayout()
    if new layout is worse (newE > E) then
      Undo the element placement with probability P = 1.0 − e^{−△E/T}
    else
      E ← newE
    end if
  end for
  Cool (T)
until rate of improvement < Threshold
```

2.3.2. Content Layout

When the amount of content cannot fit on a single page, content must be distributed over a set of pages using various *pagination* techniques. Similarly on a page, a paragraph must be broken into lines (*line-breaking*) and graphical objects must be scaled/cropped appropriately to fit the allocated regions. This section reviews briefly the state of the art work on the topics of line-breaking and pagination.

2.3.2.1. Line-Breaking

In order to place content on the document pages, text must be broken into individual lines of appropriate size. Line-breaks can be introduced between the words (hyphenation) or after depending on the quality criteria. The primary goal of line-breaking is that it should produce pleasing results, i.e., the spacing between the words is as uniform as possible, and that it should also reduce the number of times the text is hyphenated. Many of the line breaking algorithms employ *first-fit* approach where breakpoints for each are chosen one after another, and these line-breaks are not changed afterwards. But this approach does not take into account the effect of a line-break on the following lines and as a result can lead to a poor typesetting. In contrast, *total fit* approach [KP81] used by TEX considers each paragraph in its entirety, it evaluates set of all the possible breakpoints in the paragraph and selects a combination of them to minimize the total *badness* of breaking the paragraph into lines.

The *badness* is defined by individual lines depending on how much a line is shrunk or stretched. Moreover, TEX introduces *penalty* items representing the undesirability of a line-break when a word is hyphenated. Penalties are also added when consecutive lines end with hyphens, or tight lines that occur next to loose ones. TEX then counts the total demerit of the paragraph by calculating the sums of squares of the badness values (including penalties) of the result lines and finds the break points which minimize the demerit value. It can be observed that when a paragraph has n possible breakpoints, there are 2^n ways to break the paragraph into lines. However this exponential problem is tackled using *dynamic programming* which restricts the complexity of the problem to $O(n^2)$.

2.3.2.2. Pagination

Given the content, there are two ways to fit the content on screen; either to use scrollable windows or pages. The studies have shown that reading an on-screen document via scrolling mechanism shifts the focus from reading and increases mental load [WNA08]. In contrast, paging is faster than scrolling [DK97]. Distributing the content into pages helps readers in organizing the information mentally and they can recall better [PRT97] when compared to scrolled documents.

Pagination refers to dividing a document into a set discrete pages in such a way the each floating object is positioned close to where it is referenced first [Chi82]. Once the content

have been assigned to pages, layout within each page can be determined independently. Goals of pagination have been summarized by Brüggemann-Klein et.al. [BKKW98] in the following. They use a document model which uses several streams to represent content. Within each stream, all objects are of the same type (e.g., text, figures, footnotes).

Sequence: The order of objects within each stream should be preserved in the page layout.

Widows and Orphans: It is desired that the pagination avoids *widows* (last line of a paragraph printed at the top of a page) and *orphans* (first line of a paragraph printed at the bottom of a page). Widows and orphans in the text formatting are considered to be unpleasant and disturbing to the reader. Therefore, they should be eliminated.

Page Height: Each page should be perfectly full. Full pages look better than the ones containing large empty spaces inside. Moreover, it also reduces the total number of pages needed for document.

Floating Objects: Floating objects (e.g., figures, tables) should be placed in such a sequence that each floating object appears close to but not before, its reference.

Grouping and Separation Place the related items on the same page or on facing pages if they are to be compared. When required, it should also be possible to place the objects on the separate pages.

When only text is present in the document, a good pagination can be obtained by slightly under-filling or over-filling some pages to improve the overall quality of layout. But when floating objects are also included choosing the best pagination among all of the possible candidates becomes an exponential problem. Plass [Pla81] showed that computational complexity of optimal page breaking is dependent on *badness function*. Some badness functions can be solved in polynomial time, while the other measures can turn page-breaking problem to become NP-complete. One such badness function is *Quad*() which counts the page differences between each figure and all of its references. Plass proved that finding a pagination which minimizes *Quad* function is NP-complete problem which is computationally intractable. Therefore, instead of finding the best pagination, most of the pagination algorithms in practice use local optimization methods.

TEX breaks lines into pages by calculating badness ratings and penalties in the similar fashion it does for breaking paragraph into lines. Page breaking in TEX is done via a *first fit* algorithm [Knu86]. In a single pass, first-fit approach makes pages one at a time,

placing each figure in the document on the first page where it fits following its citation; there is no look-ahead mechanism to see how one page break would affect the next ones. Thus the approach cannot be guaranteed to find optimum breakpoints for the pages in the entire document, and may require hand-tuning of the document content. One reason why global optimization methods for page breaking [Pla81] were not implemented by TEX was that the computers in the early 80s did not have enough main memory to hold many pages.

Global pagination methods, on the other hand, make several passes over the input to improve pagination quality. To make the pagination computation tractable, they use restricted versions of badness functions. Often the global page-breaking optimization techniques [BKKW98, JLS+03b] involve dynamic programming approach—building up an optimal solution bottom-up by combining optimal solutions of sub-problems into higher-level solutions—a technique used for line-breaking in TEX. One drawback of global pagination methods is that they are computationally expensive and more suitable for offline purposes than online.

Another specific area of research is "Yellow Pages" pagination where the objective is to position advertisements and text-segments on a sequences of pages so as to minimize the total number of pages and various other layout constraints. The proposed solutions in this domain use constraint satisfaction techniques [GNG96] or simulated annealing [JMPS97]. These algorithms are off-line. However, it is unclear how yellow pages pagination algorithms can be directly applied to generic documents.

2.3.3. Annotation Layout

Interactive multimedia document systems present information through combined use of various media components e.g., text and graphics. The efficiency of such multimedia documents relies on a smooth integration and coordination of visual and textual elements. The dual coding theory [CP86] suggests that humans posses two independent processing systems—one for visual and the other for verbal elements. Using two channels, more material can be conveyed, but their content has to be integrated mentally. Thus, anatomic text books [MM00], anatomic atlases [SPP01, Gra18], pictorial dictionaries [PS79, Doc02], technical manuals and other books present the subject matter using pictures along with tightly integrated text.

Human illustrators employ a number of techniques to integrate textual and visual information: textual annotations in the form of *labels*, *legends*, and *figure captions*. Textual labels can either overlay their co-referential visual objects (*internal labeling*) or are arranged on the background (*external labeling*). Additional graphical objects (*anchor points*, *connecting lines*) might emphasize the co-referential relation between external labels and their associated visual objects. In practice, hand-made illustrations employ a variety of different labeling styles for internal and external labels. The choice of label layout styles reflects both space restrictions imposed by the overall layout as well as subjective preferences of illustrators.

The idea of embedding illustrative material in interactive documents is very promising. First of all the users can benefit from 3D visualization. They can explore the objects from different viewing angles which would make it easier to understand complex spatial configurations [RBF+02, PPS99]. With search or click-object options users can have the required information at hands. Moreover, the integration of textual annotations presented in labels may further increase the learning efficiency of interactive visualizations.

However, there exist very few guidelines for illustrators how to place labels in scientific illustrations except a few [Hod03, T+05]) and majority of learning and instructional material still employs static illustrations, as the depicted objects are carefully drawn by hand to convey the intended function and due to the complexity of the label layout problem. But in many technical and scientific domains the required resources are already available and can be exploited in on-line tutoring applications. Moreover, several research groups in computer graphics focus on those illustration techniques which are frequently used in technical and scientific illustrations (e.g., transparency [DWE02], cutaways [DWE03], explosion diagrams[APH+03]). In this section we review some of the progress that has been made towards label placement in graphics driven systems.

2.3.3.1. Problem Statement

Labeling problem for dynamic environments is stated as follows: *given a set of graphical objects and their associated annotations, find a layout that fulfills the following requirements*([Imh75, FP99]):

Readability: Labels do not overlap each other and other key features in the visualization.

Unambiguity: Labels should clearly refer to their associated objects to avoid *referential ambiguity*. Referential ambiguity arises when the readers cannot maintain *association* between annotations and their corresponding objects. This occurs due to poor placement of anchor points and labels.

Aesthetics: Layouts should display symmetry and reduce the visual clutter. For external labeling following requirements could be met to improve the layouts:

- Align the labels together and with respect to graphical objects (*brings the symmetry*),
- Eliminate line crossings (*reduces the visual clutter*),
- Place labels near to their objects (*keeps the connecting lines short and reduces the chance of referential ambiguity*), and
- Place anchors at the salient locations of objects (*helps in object recognition*).

Interactivity: Labeling should be computed in real-time to allow the dynamic user interaction with the 3D visualization. Moreover, there should be no heavy visual discontinuities in layout over the time. In interactive 3D systems, computing label layouts solely from individual frames would result in flickering and would distract the user. Frame coherence is needed to bring smooth transitions in layouts.

Compaction: Reduce the layout area. The annotated illustration must fit within the alloted space. The other less strict demand may be that of labeling as many objects as possible (*label maximization*) in the given area.

The criteria describe the generalized requirements needed to perform the labeling task. The automated layouts should also display the symmetry that is often seen in manual layouts.

2.3.3.2. Label Placement in Cartography

Label layout has received much attention in cartography, where text labels are placed in maps while avoiding overlaps with map symbols and other labels. Manual label placement is a very tedious task [CJ90] since a map may contain thousands of locations which need labels. Therefore, automated label placement solutions are required to ease the burden.

Traditional map labeling problem is divided into three tasks: *area feature labeling* (names on oceans and countries), line feature labeling (names curved along rivers and roads),

and point feature labeling (city names or mountain peaks). However, labeling problem is independent of shape of features and can be unified for point features [KT98]. Therefore, research towards automated label placement in maps focuses on *point-feature label placement* (PFLP). An instance of PFLP problem consists of a set of point features, labels, and mapping from labels to point features. The task is to place labels adjacent to the point features, so that no or very few label overlaps take place.

The PFLP problem can be considered as a *combinatorial optimization* problem, for which a solution is chosen from among a *finite* set of possible solutions. Search space for PFLP consists of a set of possible label positions for each point feature. A typical search space includes eight possible label positions for a point feature (see Figure 2.3). The rectangles in the figure indicate the regions in which the labels can be placed. An optimization procedure looks into the search space to find alternative positions for a label while improving the solution. However, searching the globally optimal solution for map labeling has been proved to be an *NP-hard problem* [FW91, MS91] since the placement of a single label can have a global effect on label overlaps in the entire map. Therefore, a number of heuristics and approximation algorithms have been developed to reduce the computational complexity. The local optimization techniques improve the label layout based upon the information available in the neighborhood rather than the entire search space. A comparative empirical study of PFLP optimization methods (Exhaustive Search, Greedy Algorithms, Gradient Descent, Simulated Annealing and Linear Programming) is given by Christensen et.al. in [CMS95].

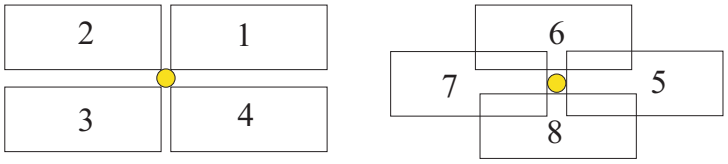


Figure 2.3.: Possible positions for a label. A label is set up at one of the eight positions shown in the picture. Lower values indicate more preferred positions.

In the past, map labeling applications were designed to create large sized static maps for print-purpose with no time restriction. Recently, researchers are exploring techniques for generating *dynamic screen maps*. In contrast to static maps, screen maps are of smaller size due to lower screen resolution and display size. Dynamic screen maps must be generated at interactive rates and should support region selection, scrolling, zooming, and integration of several layers of information, often requiring an update in label layout [PGP03, DPP03]. They might also consider user-specific requirements within the map generation [AS01].

Many dynamic scenes (both 3D and their 2D projections) require labeling for objects that are not fixed but can move relative to each other and relative to an observer's position. Kämpke [Käm03] extends static label placement to dynamic scenes on radar screen. His approach uses *artificial hysteresis* [SSS97] to reduce the label movements that are attached to their moving objects.

Though automated PFLP techniques were mainly developed for cartographic applications, they have been also adapted for applications in other areas such as geographic information systems, data visualization, graphical interfaces and diagrams where point objects are tagged with labels. But except a few methods such as in [BKSW04], it still remains unclear how the classical cartographic methods could be extended to meet the dynamic requirements of interactive visualizations. Therefore, the annotation placement research in interactive illustrations has taken a different route than cartographic name placement.

2.3.3.3. Interactive 2D/3D Illustration

Effective automated label placement strategies lack in both 2D and 3D illustration systems, and most of the systems present text for selected objects without addressing the layout problem. Online visual dictionaries such as [MW08] use static illustrations. BrainInfo Atlas [Bra00], Sobotta Atlas CD-ROM Version and InnerBody [Inn03] present annotations on 2D images in the form of tooltips (see Figure 2.4(a) and 2.4(b)). Lens Explorer designed for creating exploding diagrams for 3D models interactively also uses tooltip mechanism to display annotations. In AnatomyBrowser [GKU+98], labels for the selected objects are added into the illustrations in a pre-determined order. Crossing between connecting lines often arise (see Figure 2.4(c)). Other systems like VOXEL-MAN [VM] and [LAS04] rely on user interaction to achieve an appealing label layout [LAS04, RSHS03].

These examples highlight the lack of of automated label layout techniques in various interactive illustration systems. This is not only in the case of interactive illustrations, but also with non-interactive illustrations where human illustrators painstakingly annotate objects. Integration of text in 3D illustrations is even more difficult as the visualization changes over the time due to user interaction. Manual label placement in 3D illustrations would be impractical as certain object may appear or disappear and shift their positions on the display, thus making the label layout obsolete. In contrast, an automated solution could improve the readability of illustration by adjusting the labeling as the scene changes.

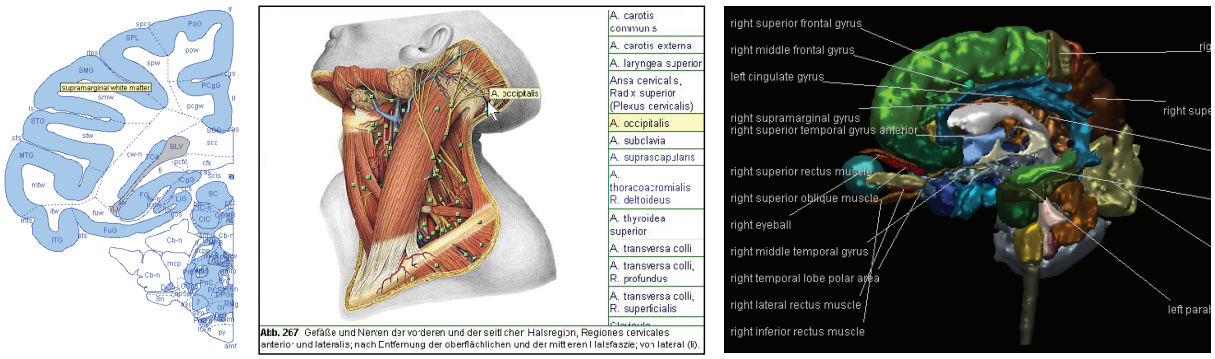


Figure 2.4.: (a) BrainInfo Atlas [Bra00]. **(b)** Sobotta Atlas CD-ROM Version. **(c)** Anatomy-Browser [GKU+98].

Automatic label placement in scientific illustrations has received little attention from the industrial or research community and the labeling problem in visualization-based systems has stayed at low priority. Therefore, very simple solutions exist which either don't demonstrate a tight integration of labels with the graphics or lack the quality which is seen in hand-made layouts. Preim's Zoom Illustrator [PRS97] places text and 3D graphics in separate panels. The text positions remain static during user interaction while the connecting lines are updated. Intersections between connecting lines are resolved by repositioning annotations on user's request. The system uses mesh reduction algorithm [PR98] to generate multiple anchor points for each object. Illustrative Shadows which is aimed towards the dynamic exploration of 3D models inserts annotations at fixed locations. It does not deal with the label layout problem at all. Hartmann's Floating Labels [HAS04] employed dynamic potential forces to determine external labeling for geometric models. Attractive forces are established between textual labels and corresponding objects, and repulsive forces are established (i) between labels and other objects, and (ii) between all labels. Both kinds of forces are then summed up to find the label positions. However, their system is computationally expensive and does not work at interactive rates.

Goetzelmann [Göt04] developed algorithms to place text labels directly on corresponding 2D projections of 3D objects on screen without using connecting lines. His labeling approach is similar to area feature labeling in maps where the label style and orientation follow the shape of object. The limitation of his approach is that it only deals with internal labels. If the projections of have big dents, contain holes or have thin and have a shape like a "U" or "O", internal labels may cause association problem and can be difficult to read. In addition, if a complex geometric model consists of several tiny objects, it may not be

feasible to place labels inside the objects. In contrast the approach of Chigona [CSRS03] avoids directly setting up labels over the objects. Their system Talking Shadows places the labels outside the objects but inside the object shadows (see Figure 2.6) . Label overlapping is allowed in their system which does not address the label layout problem.

Figure 2.5.: Internal Labeling [Göt04]. Skeletons of objects' projections are computed, and then smoothly curved for laying out internal labels.

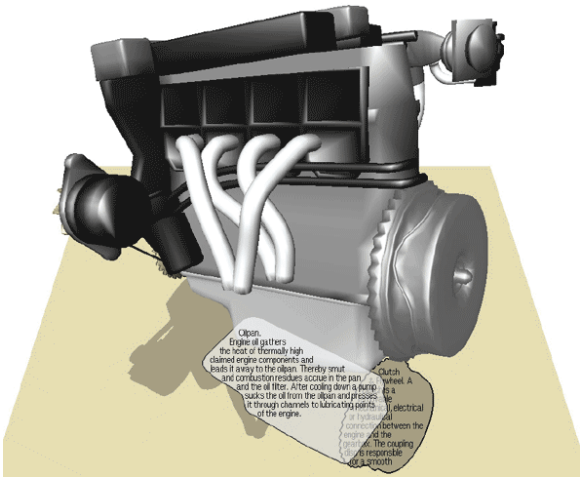


Figure 2.6.: Talking Shadows [CSRS03].

Recently, in AR/VR the term *view management* is used for a more general, but related problem: the smooth integration of additional 2D information (images, texts, annotations) into the view plane [BFH01, AF03]. Previous work [FMHS93, TMR+97, YNA99] related to adding annotations in AR/VR environment didn't take care of *visibility constraint*. This often led to situations where annotations severely occluded other features that they referred to. Bell's approach [BFH01, BFH02] uses *Empty Space Management* [BF00] to locate empty rectangles for placing internal and external annotations (see Figure 2.7). It computes axis-aligned rectangular approximations of object projections on view plane which can cause errors in annotation placement.

Another area where labeling has its application is *Information visualization* (InfoVis). In addition to graphical information, the visual presentations often accompany text labels to describe the abstract information. Label placement is a common problem in all kinds of InfoVis systems . For example, *molecular graphics* (e.g., circular maps, protein structure wheels) [Bri96] are used to display DNA and RNA genetic sequences and protein structure. Usually these visualizations are flooded with a large number of text labels which are unreadable due to the label overlapping.

Many InfoVis systems require annotation of point features, and demand minimization of number of label overlaps. For this purpose, cartographic PFLP (Point Feature Label Placement) methods can be used. However, PFLP may be ineffective as these system often have large amount of data. Therefore, many techniques such as *focus+context* [NPMK97, SB92] and *dynamic query filtering* [AS94] have been proposed which shrink/filter less important objects and reduce the information on screen to create room for labels. LifeLines project [PMS+98] uses *focus circle* to simplify the labeling task through filtering out information. To retain all graphical information on screen and provide text labels when required is demonstrated in [FP99] through *excentric labeling* (external labeling) for labeling localized objects in the 2D scatter plot. When the mouse cursor is located over an area for a while, the point features in the focus circle are annotated via axis-aligned external labels (see Figure 2.8). [LSC08]

Figure 2.7.: View management for virtual and augmented reality [BF00].

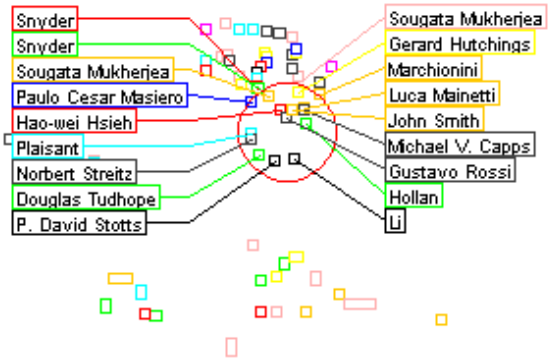


Figure 2.8.: Excentric labeling [FP99]. Given the point features (like in left picture), it uses external labels to annotate points in focus circle.

External labeling problem has similar requirements to those of graph drawing [DETT99]. One can think of external label layout in terms of *graph layout*: both anchor points and label positions as vertices, and connecting lines as edges between vertices. In visualization, the purpose of graph layout is to draw nodes and edges, while the purpose of external labeling is to draw labels and connecting lines. Both external labeling and graph drawing try to improve the layout readability—minimize line crossings, reduce length of connecting lines, enhance symmetry— and fit the layout in the restricted space. Thus most of the criteria of graph layouts can be also used to evaluate external label layouts. Traditionally, most of the graph drawing algorithms only deal vertices as zero-size points, but there has been an ongoing research in non-uniform vertices of arbitrary size and shape [HK02]. When graphs have non-uniform vertices, they become more similar to external labeling as the

sizes of graph vertices can be set according to sizes of anchor points and labels. However, in case of external labeling more is required (e.g., anchor points would be computed that must overlay the objects, and once anchor points are set up, the repositioning would be allowed for labels only. The labels would avoid obscuring the other labels as well as the graphical features in the image).

2.4. Document Browsing

2.4.1. Interaction

Modern multimedia documents should increase the readers' interest by actively engaging them in learning tasks. The readers support be supported with continuous interaction mechanism to control and operate the presentation. Traditional user interfaces, mice, keyboards, windows, menus and short-cut icons provide useful mechanism for browsing digital documents. Now we have 3D pointing devices, haptic and multisensory input/out devices that the modern 3D illustrated documents can now explore to improve human-computer interaction. Interaction tasks in multimedia documents can be classified to two broad categories (adapted from [BKLP04]):

- Navigation,
- Selection and Manipulation

Navigation: Digital documents can contain large information spaces and can be linked indefinitely, therefore, navigation becomes an important task in document browsing. The objectives of navigation in a document are exploration and searching of information. There are several navigation interaction techniques for different applications but in the following only the topics within the scope of dissertation are discussed:

Basic Navigation Techniques: A *hyperlink* in a document is a basic navigation element to other objects in the same document or to another document. Standard web browsers support back and forward buttons, history and bookmarks etc. to aid the navigation tasks in previously visited web pages. Many document readers such as Adobe Acrobat Reader feature Table of Contents as bookmarks for different parts of the document. Standard navigation techniques provide location information within the document. What chapter, section or subsection etc. are you in? Documents can be either displayed using

scroll mechanism, or divided into discrete pages or use scrolled pages. Electronic document browsers these days support search facility but the searching in illustrated media is limited as majority of illustrated documents contain static illustrations. Another concern is that often the illustrations and explanatory text in the presentation are not coupled together. Practically all interactive 3D illustration systems present text and illustration in their own panels. This mean the user navigate back and forth from one panel to another which can impede learning. TEXT ILLUSTRATOR links long-text and geometric models [SS99]. Navigation in the text panel (scrolling or selecting) triggers navigation in the visualization and vice versa. However, TEXT ILLUSTRATOR does not present the textual information and visualization in separate panels. This is unlike visual dictionaries in traditional media where the document contains both words and annotated pictures are seamlessly integrated together in the same surface.

Zooming Interfaces: Browsing multi-page high-resolution document on small screens could be difficult for the reader. One solution is zooming interfaces that present the document at different levels of detail. Users can zoom in/out the objects of their interest and pan the page across the screen. Examples of early zooming interface are PAD++ [BH95] and PAD++ HTML zooming browser [BHS+96]. The latter extended HTML to support multiple levels of zooming within a single web page. This extension MULTI-SCALE MARKUP LANGUAGE (MSML) introduced `<meta>` tags to create a multiscale hierarchical outline of the contents. However the goal of zoomable HTML browser was to produce interactive web pages which can be zoomed-in and zoomed-out rather than adapting the content of the pages according to user goals or interests. EROL et.al. [EKJ06] demonstrate MULTIMEDIA THUMBNAILS(MMNail), a computer-generated guided-tours of documents. MMNail animates document pages, zoom into and pan over the most important elements of the document such as titles and figures, automatically. Futhurmore, it utilizes audio channel of the portable device to describe textual information. The system doesn't deal with the adaptation of document. A similar system SMARTNAILS [BSM04] which creates document thumbnails for a particular screen size using cropped/scaled images and (truncated) text segments. It uses image analysis and text analysis techniques to extract key information from the scanned documents. Though SMARTNAILS does the reformatting of the content on screen, the approach is limited to creating thumbnails only. THE INSTANT BOOKPLEX [BPGN04] is another system to use graphical zooming interfaces for helping the user search through a document collection.

Focus & Context: Instead of zooming the whole presentation area as in zooming interfaces, focus & context techniques combine detail (*focus*) and overview (*context*) seamlessly into the same visualization. The Perspective Wall [MRC91] uses 3D visualization to display documents on three panels. Centered panel is used for displaying the detailed information and two perspective panels for the context (a zoom-out). In [RM93], a document lens is used to magnify the regions of interest in 3D while remaining in the context. Such focus & context techniques can be regarded as *visualization-based* because they only magnify the visualization and not adapt the content. The visualization-based focus & context methods may not work well on small displays with small resolution and limited capabilities. In this case rather *content-based* focus & context techniques are more useful which magnify the actual content. For example in [BGMP01], a web document is presented as a summary; depending on user's interest the relevant regions can be expanded while remaining the context.

Selection and Manipulation: Basic interaction techniques in standard web browsers are point & click, selecting content items, typing and copy & paste functions. In 3D visualizations typical selecting and manipulation techniques involve touching 3D objects, rotating, positioning or scaling them in parts or as whole [BKLP01]. Depending on the context, selection tasks performed by the clients can also lead to the document personalization, explicitly or implicitly.

2.4.2. Personalization

In digital media the intentions and requirements of the readers are dynamic. One goal of the interactive multimedia documents should be to provide their information recipients with what they want and how they want. Document personalization involves determining the user's requirements during interaction with the user to deliver the appropriate content. This means selecting relevant content and generating a presentation tailor made to the user's preferences. Personalization is crucial in World Wide Web as it allows authors to improve the experience of large group of information recipients. There are three broad categories of personalization models, *content-based filtering*, *rule-based filtering* and *customization*. Content-based techniques filter the information based on what the user has favored in the past. In *collaborative filtering*, the recipients are served with the information by combining their own preferences with the choices made by other users [col]. An example of collaborative content-based filtering is `Amazon.com`. Rule-based

personalization refers a technique that matches the user-profiles to the content profile to select the content based on inference rules [Ins00, HS04]. A user profile may include user's personal information (gender, age), role, geographic location, type of device to access documents their interest and preferences. Content profiles may store descriptive, semantic and structural information and even presentation attributes. By applying ("if this - then that") rules on the user profile, the system can then look-up in the content profile the relevant content for a particular type of user. Many search engines and companies offer rule-based personalization for their websites.

Customization, on the other hand, is done by the individual users to customize the user interface according their own needs and taste. The customization feature list can include:

- customizing physical attributes of information elements, such as typeface and color etc. ,
- layout customization,
- adding annotations, notes and
- highlighting information elements.

Current web browsers give no support for personalizing documents as such other than customizing font size and zooming. Some websites allow the customers to choose from built-in templates and customize their own templates. But often is the case that users are given no option of customizing design features of documents.

2.5. Discussion

Effective presentation of interactive multimedia documents those can be viewed on many different devices and respect user-specified preferences pose many challenges for the design of automatic layout algorithms. On the one hand, authors should be able to provide well-balanced layout templates for common displays. On the other hand, the authors cannot bear the burden to foresee the requirements of all possible use cases.

This chapter reviewed the problem of creating adaptive documents. In section 2.2, we discussed that content representation is crucial to authoring adaptive documents because adapting the layout for different viewing scenarios requires adaptation in content as well.

For that, authoring alternate versions of content, and choosing the best version during the layout process is a good solution.

As discussed in section 2.3.1, modern multimedia documents usually incorporate static layout designs and lack effective layout strategies. Early research in document layout mainly focused on text formatting and produced static layout aimed at printing. Static layouts are not adequate in web environment which requires dynamic behavior. In web, markup languages and style sheets do provide a way to separate content from layout, but there is no convenient way to write adaptive content and suggest how the document layout should be adapted. We aim to tackle the document layout problem with a new approach.

Another problem is producing and integrating interactive illustrated material in digital documents. Label placement in 3D illustrations still remains unexplored and many systems either employ manual label placement or provide annotations when the objects are mouse selected. The state of the art automated labeling techniques are either limited in scope or do not meet the labeling criteria described in section 2.3.3.1. In our work we shall present a new approach towards automated external labeling for 3D illustrations that is purely aimed towards increasing the quality and the usability of labeling illustrations. It can create a variety of external labeling styles which can be combined with internal labels. The main source of inspiration behind the work is hand-made illustrations where manual labeling enriches the pictorial information in a pleasant way.

The digital documents should provide the users with high degree of interaction to accomplish learning tasks (cf. section 2.4). We believe that it should be possible for the users to interact with not only the document itself but also with all information elements contained inside, and personalize/customize them according to their needs.

Chapter 3

Conceptual Framework

Traditional media conveys information with physically static content and presentation. The static nature of documents is unavoidable in physical environments. In contrast, the digital documents enable the publishers to present information dynamically to a wide variety of clients connected to network via different devices. In this scenario, a digital document is considered continuous in time, reflecting the contextual requirements of client requesting information. Since manual work would not be fast enough to meet these requirements in real-time, automated techniques are needed to generate documents that are tailored to the needs of users.

Automated generation of digital documents is a difficult problem which encompasses various subtasks such as determining what information to present (*content generation & selection*), how to present it (*layout formatting*), and how the users shall view or interact with it (*interaction*). In this thesis, we focus on the development of automated techniques for document layout formatting. Within the scope of thesis, it is assumed that the content is either provided or manually authored using the tools discussed in chapter 4

Given the content, the goal of an automated layout process is to determine the positions and sizes of graphical objects on the page according to a set of criteria. As discussed in section 1.2, such a system should support the human designers to describe a set of dynamic design solutions that can reflect the continuous changes in both the readers's intention and the content.

In this chapter, we discuss the issues regarding the personalized document creation. A generic adaptive document layout framework is proposed for creating dynamic design solutions in digital documents. In this framework, a design solution is computed by solving a set of required and optional constraints imposed by the designers, content authors and the viewers.

3.1. Problems and Requirements

Up to now, a lot of research and development has addressed the automated multimedia documents. But, there has been no generic solution for adapting the presentation layout to a variety of viewing conditions. Current approaches in this regard are either non-automated or limited in scope. In order to analyze the problem, we discuss the process of document creation in traditional and digital realm. Both the traditional and digital media must deal with three issues for publishing documents—content management, layout formatting, and viewing (interaction) [Ish03]. These issues are addressed by three types of users:

- *Author:* An author prepares content to be communicated and performs organization and association of the content.
- *Designer:* A designer creates visual layout designs to communicate the intended content.
- *Client:* A client requests information. The information is formatted and presented to the client in the form of document.

But, there are some basic differences between traditional and digital media. In digital media, information content and intentions of readers change continuously as well as the output medium. Furthermore, the digital media requires that information is formatted appropriately to the display device. These dynamic requirements of the digital media make the designing process difficult. Since the content is available only at run time and accessed at different display devices, the document design cannot be determined in advance. Unlike a traditional design solution which is fixed, a dynamic design solution is a continuous process of manipulating design and reflecting the changing scenario. Manual work employing human designer to design information continuously would not be fast enough to meet the requirements of a dynamic environment. One option is to use *templates* that can be filled with the content by an automated process. But if the design scheme

solely relies on preformatted templates, it can severely limit the range of possible design solutions. Instead, what we wish for is an automated design process that could create flexible design solutions for each individual audience. However a computer-based design system should execute design on behalf of human designer. Human factor is important because document layout should take care of communication goal, aesthetics and cognitive principles. Furthermore, as often is the case with designers they are very concerned about how the information would be presented and want to control each and every aspect of design manipulation.

This scenario raises a few questions. How can designers describe design solutions that can capture changes in information and reader's intention? How can designer conceptualize design problem in digital media as a continuous problem instead of a series of discrete problems? Based on this, it is useful to develop a design system whose behavior could be specified by the human designer. A designer's role in such a framework is to construct *initial design patterns* in the system and specify how they can be adapted to solve the dynamic design problems. The initial design patterns may consist of a range of adaptable templates specifying layout constraints. These templates could be continuously modified to accommodate changes in context while maintaining the overall design at the same time.

The use of templates in the digital media can be justified on the basis of three basic core design principles. These principles are, *Effective Separation*, *Flexible Presentation* and *Reusability* [Web]. The goal of first principle is to differentiate between content and style. Templates are used to provide this separation. The second principle refers to the ability of the system to change the layout of the page without affecting the underlying structure while putting as little burden as possible on the human designer. The principle of reusability is needed because it allows the designers to share the template designs with others. It is important that the other users should be able to change the design according to the likeness and needs of their projects.

What is actually needed in this scenario is to find a design solution that satisfies the layout constraints imposed by the designer, content author and the viewer. For constraints specified through the content, a way is needed to extract the semantic values in the content, such as figures references, relevance to the topic, etc. and translate them to layout parameters. Ultimately, this would require us to build a layout mechanism in the system which can generate design on its own, where the final appearance of the document is a negotiation of all constraints.

3.2. Adaptive Document Layout Framework

A generic adaptive document layout framework is proposed that is shown in figure 3.1. The framework architecture resembles the general document design process—content, layout, and interaction—where each role is accomplished by respective user.

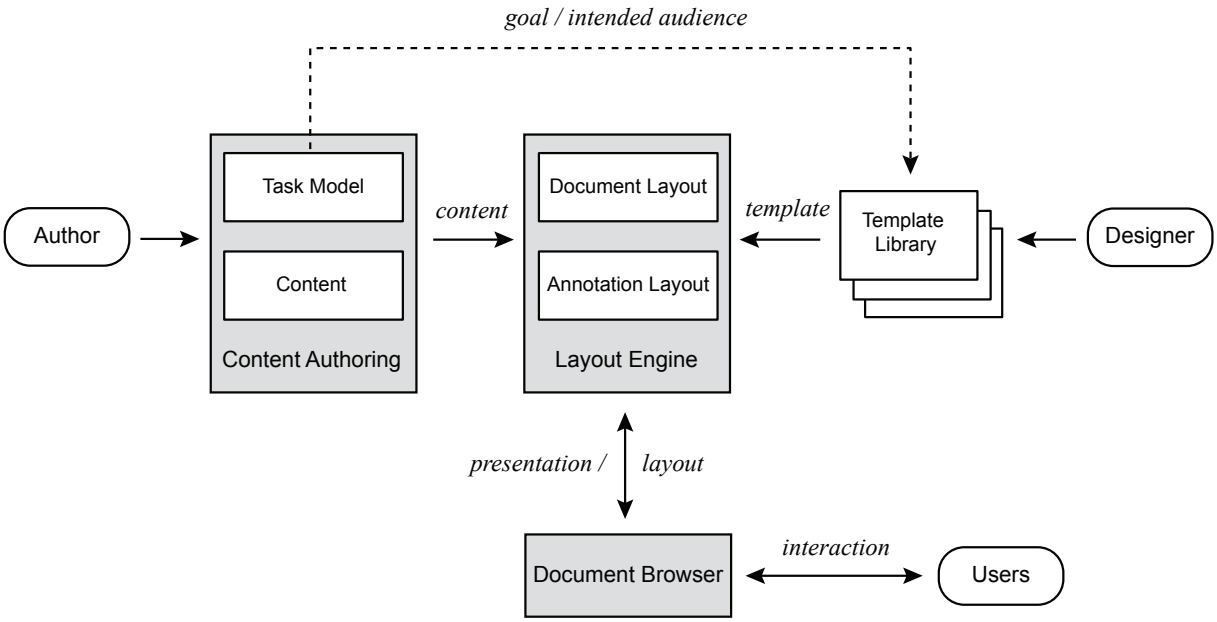


Figure 3.1.: Architecture of adaptive document layout framework.

At the core of adaptive document framework is the content and style separation. Authors and designers can prepare content and presentation styles independent of each other. However the designer must pay attention to the nature of information, the goals of communication, and types of intended audience. As a result of design problem analysis, the designer directs the design process by creating a set of multiple templates. On the authoring side, authors prepare document contents by tagging individual content items such as text, static illustrations, animations, and interactive 3D representations. The content is stored in XML tree structures that enables hierarchical organization of data. Authors can provide alternative versions of any type content which can appear in the document, such as text, images or styles. The authors can also create task models [AHFS08] to store the guidelines about content and layout for particular learning tasks. They can specify for each task which template from the library is preferable for the content associated with that task. During the document delivery process, the guidelines in the task model are used to influence the document design process. The details about authoring design and content process are outlined in chapter 4.

The heart of the framework is automated layout process that generates dynamic design solutions. The layout engine which consists of two major parts—document layout and annotation layout. The process of automated layout adaptation of online documents is outlined in figure 3.2. The user requests the information and the adaptive document system produces the document. The *content and template selection* stage chooses the content and suitable templates to generate the document. *Content selection* component retrieves the content and filters the information depending upon the context. The output of the content selection layer is refined content and structural information of the content. The content selection process is further guided by the task models that can also specify presentation guidelines. The *template selection* component determines the appropriate set of candidate templates from template library. The layout engine evaluates all design templates against the selected content and the viewing device. For the task at hand, the system determines the order in which the selected content needs to be presented. The ordering of the content in the presentation depends on the semantic relations within the content and communication goal. Moreover the ordering of the various media can also be constrained by the template and pagination task. In other words it is negotiation among several tasks.

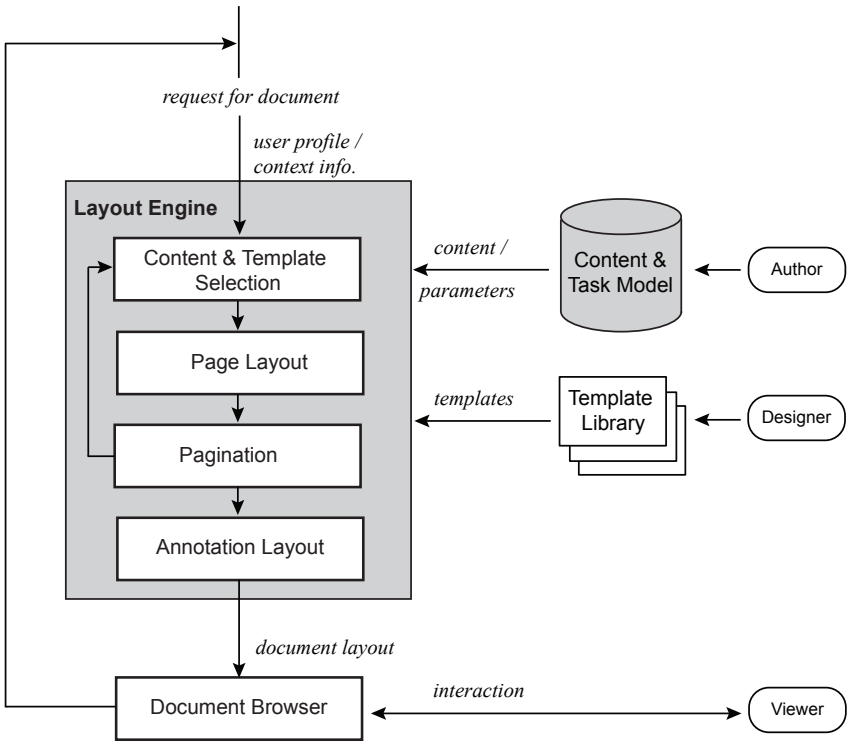


Figure 3.2.: Adaptive document layout process.

During the *page layout* step, the layout engine adapts the templates to the screen size and flows the content into template regions to find how good the content fits in the template. This information is passed to the *pagination* stage that calculates optimal sequence of templates for paginating the documents. More details of first three stages of layout engine are presented in chapter 5.

For illustrated figures, the *annotation layout* stage labels the objects with their names to establish co-referential relation between graphics and text (see chapter 6). Powerful annotation layout techniques are presented that deal with the internal and external placement of annotations in the graphics. A number of layout styles are implemented which are based on functional and aesthetic attributes in hand-made illustrations. The annotation algorithms adapt the illustration to the size allocated by the layout engine. Annotations are added to the illustration only for the relevant objects in order to reduce visual clutter.

At client side, the browsing interface lets the audience to have an interaction with the document and interactive illustrations. Depending on the interaction (e.g., resizing browser, selecting a content element, or rotating a 3D model in the visualization), browsing tool allows the viewers to specify contextual requirements.

In any case, it is important that the performance of layout engine is good enough to give an interactive experience to the users. It is also essential that the document presentation is smooth and consistent over time. For example, each time a browser is resized, the document layout must be recomputed. Similarly, when the user rotates or zooms within a 3D illustrations, the annotation layout needs to be refreshed. As a result of layout update, the layout elements might shift to different positions or may either reappear or disappear. Such changes between successive frames of presentation can draw the users' attention towards potentially irrelevant objects and spoil the reading experience. Interactive documents which provide the information in temporal frame should try to avoid abrupt appearances, luminance flicker and rapid motion. If it is not possible to completely avoid visual discontinuity, its effect should be reduced in the presentation.

3.3. Summary

This chapter examined the spatial layout problem in the domain of multimedia documents. The current layout implementations in the document browsers rely on static designs and

have a number of serious limitations in adapting the document to the the given context. The difficulty arises because of the dynamic nature of the digital realm where the content and user requirements are updated frequently. The designers have to create designs for content and viewing scenarios both of which are unknown at the time of designing. This leaves us with the only option that the design process is executed by an automated process at the client's computer. The challenge is to generate design solutions that could meet the intentions of authors and information recipients continuously over time.

A new approach to generate automated document layout is presented that would determine the size and position of text and graphics in the presentation. In this approach, authors can create alternate versions of content and designers can specify loosely how the content should be displayed. The actual document formatting is handled by a computational layout engine which is able to generate design solutions continuously for a wide variety of viewing scenarios. To ensure that the presentation retains the designer-specified rules, the layout designs take into account the templates. Another novel aspect of the work is annotation layout approach which deals with the adaptation of annotated 2D/3D illustrations embedded within the document.

The first step towards implementing this framework is authoring content and designing templates which will be discussed in the next chapter.

Chapter 4

Authoring Adaptive Documents

As discussed in section 2.3.1, creating a presentation layout is a complex problem that requires human skill to design. Human input is an important factor when designing either traditional or digital media despite the differences in both media. Though, the use of ready-made templates designed by professional designers is prevalent in traditional media as well as in World Wide Web, such design solutions are static and will have limited flexibility when it comes to the adaptation of layout based on changing context and viewing scenarios. Usage of static template may result in a document that has superior layout quality at the desktop screen, but looks poor when viewed at PDA device. This is an inherent problem with the template-based approach because a template basically reduces the degree of freedom of design to limited number of solutions. A static template is just a fixed solution specified by the designer. In a dynamic environment, a design scheme that relies only on filling of static templates with new content without taking into consideration how well the content fits to the screen will face the problem of producing poor presentation layout. Therefore, an elegant compromise is required, where the designers can express essential design qualities using templates that could be adapted over the time by an automated process to match the contextual requirements.

Consequently, the adaption of content and presentation must be acknowledged. Content items should be specified at multiple levels of details or transcoded into multiple resolutions or modalities. Multiple versions of content would enable to generate different versions of documents and/or to optimally match the capabilities of rendering devices. Similarly, designers may wish to design a sequence of adaptable templates to cover a range screen

sizes and achieve various communication tasks. The adaptation of both the content and the design would allow the system to generate adaptive documents in dynamic environment.

Since this thesis aims at the adaptive documents with a particular focus on the automated layout, the layout engine needs to consider not only template adaptation but also content adaptation. With the varying viewing scenarios, it is important that the content items are tailored to fulfill the layout requirements and provide appropriate information to satisfy users' information needs. Ideally, the task of content adaptation should be automatically performed. Nevertheless, an extraordinary diversity among users, their goals and backgrounds make automated authoring of content a very challenging task. The difficulty of the problem is highlighted by the fact that there are a number of researchers within adaptive hypermedia and information retrieval systems working on the problem of exploiting contextual information to deliver what clients' requirements. However, the topic of automated content adaptation driven by the individual user's behaviour is beyond the scope of the thesis. The approach presented in this work assumes that the document content is given in the format understood by the framework, or can be produced by authors manually. Authoring process, explained in this chapter, involves authors to create *multi-representation* content tree that is used to generate different combinations of content in the adaptive documents.

This chapter describes the authoring process for adaptive documents. Section 4.1 introduces the authoring steps to produce adaptive documents. Section 4.2 presents the multi-representation content model and then describes a system for representing the multi-representation content using XML. With each content item, authors can indicate viewing scenarios under which particular segments of content should be chosen for different users. The authoring tool for creating the adaptable layout designs of the documents is explained in section 4.3. In section 4.4, we discuss the use of *task based model* to enable the authors to generate different views of documents. Section 4.5 concludes this chapter.

4.1. Overview of Authoring Process

In the adaptive document layout framework, *authoring* refers to creating content, style and task models for the documents. Dynamic nature of the online media demands that document framework should support flexible content and presentation structures to allow automated document assembly. Authoring is a crucial step in this context as it allows

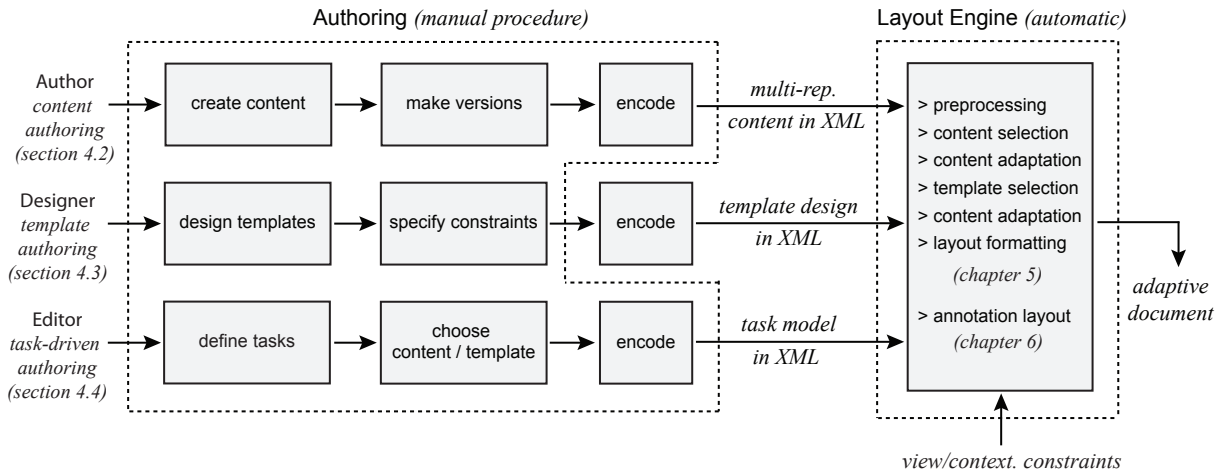


Figure 4.1.: Overview of the authoring steps in adaptive document framework.

the authors to create appropriate content pieces and specify the layout design rules, both of them can later be dynamically adapted. Authoring process comprises of three parts, namely, *content authoring*, *template authoring* and *task driven authoring*. Figure 4.1 depicts these steps in the adaptive document framework. The authoring steps are further explained in the following sections of this chapter.

Content authoring process relies on the manual effort of authors to create content. Authors prepare the document content by tagging individual items, e.g., text, static illustrations, animations, and interactive 3D representations. Next, they create different versions of document content. This is accomplished by producing multiple resolutions of content elements, choosing modalities, and defining content hierarchies. In the encoding step, authors may use an XML editor to create XML-based multi-representation of content.

In the process of *template authoring*, *designers* create design templates using a graphical user interface. Designers draw containers for text and graphic elements and specify spatial constraints, for example, optimal size and positions for elements in the layout. Design templates are encoded in XML format by the template editor and stored into a *template library*.

Task driven authoring steps allows the *editors* or *authors* to generate different views of a document. A task model stores guidelines about the document views, e.g., which content elements and design templates are most suitable for a specific task. Using a task model, the content can be fetched from a single or multiple content sources and authors can even hard-code new content for the document. A task model is also represented in XML file format.

During the layout process, task model is used by the layout engine as a preprocessing step to filter the content and establish candidate set of templates for formatting the document. This leads to appropriate selection and adaptation of content and templates with respect to the viewing and contextual constraints. In the last step, the content is synthesized into multimedia representation of a document (see chapter 5 for document layout, and chapter 6 for annotation layout).

4.2. Content Authoring

Just as different readers may require different versions of content, different layout templates may also require that an appropriate version of content is chosen based on the properties of viewing device. Consequently, the layout adaptation cannot be separated from content adaptation and vice versa. Therefore, first step towards achieving adaptive documents is designing a document specification that supports multiple representation of content. In order to design a document specification, a content representation scheme based on InfoPyramid [MSL99] is used. InfoPyramid is a scheme for representing content in multiple modalities, levels of abstraction and resolutions (see section 2.2). We adopt the same XML notation as in [Li02, SJL08] to specify overall document structure and add further attributes and nodes to it. The document content is stored in a tree structure that has two types of nodes: *content nodes* which store the actual content at the leaves of the tree, and *structural* nodes of two types, *Or* nodes that combine alternate versions of content, whereas *And* nodes explicitly group together *Or* and content nodes.

`<Rep>...</Rep>` element represents a content node which stores a single version of actual content in its child XML node. The child nodes of `<Rep>` can be, a `<img>` element for specifying image, a `<text>` for text paragraph, and `<illustration>` for interactive illustrations. The attributes in `<Rep>` element are used to list the properties related to the content, and guide the layout engine to select appropriate version of content to lay out.

The *And* node is represented by `<And>...</And>` element. The purpose of *And* node is to simply group an arbitrary number `<And>`, `<Or>` and `<Rep>` elements. The *Or* node is represented by `<Or>...</Or>` element and can contain arbitrary number of `<And>`, `<Or>` or `<Rep>` elements. *Or* node acts like a switch to provide an alternative choice of content from its child nodes. For example, an XML representation of alternate versions of an image would be:

```
<Or>
 <Rep version="full">
  <img src="imgA.jpg"/>
 </Rep>

 <Rep version="short">
  <img src="imgB.jpg"/>
 </Rep>
</Or>
```

In this example, the `version` attribute inside of `<Rep>` element points out that for the `full` version of article, `imgA.jpg` should be selected, whereas for `short` version of article, `imgB.jpg` should be chosen.

Figure 4.2 gives an example of multi-representation document content tree. XML code of the content tree is given in the figure 4.3.[1] The example shows how a combination among `<And>`, `<Or>` and `<Rep>` nodes enables authors to create multiple versions of content with multiple modalities, and resolutions as in InfoPyramid. By using the multi-representation content, the layout engine can automatically select alternate content to yield the optimal layout in different contexts.

Usage of `minPageWidth` `maxPageWidth` attributes inside `<Rep>` elements can be noticed in the figure 4.3. They specify the minimum and maximum bounds of page size that the content items should be used for. These parameters are taken into consideration during the layout process to decide which content to be chosen compared to others (see next chapter). In the XML code on line 5, `minPageWidth=480` indicates that the `teaserText` is appropriate for page size bigger than 480. The line 8 indicates that for a small-sized screen `maxPageWidth=480`, the teaser should be left empty. The values of these attributes are used by the layout engine to compute the penalties for each type of content during the layout process. The content version with the lowest penalty is placed on the document.

1 The current implementation requires that the authors prepare the multi-representation content directly in XML format by using an XML editor such as Altova XMLSpy. Future implementation should facilitate a graphical user interface, similar to [Li02], for preparing multi-representation content in graphical form and thus, generating XML representation automatically.

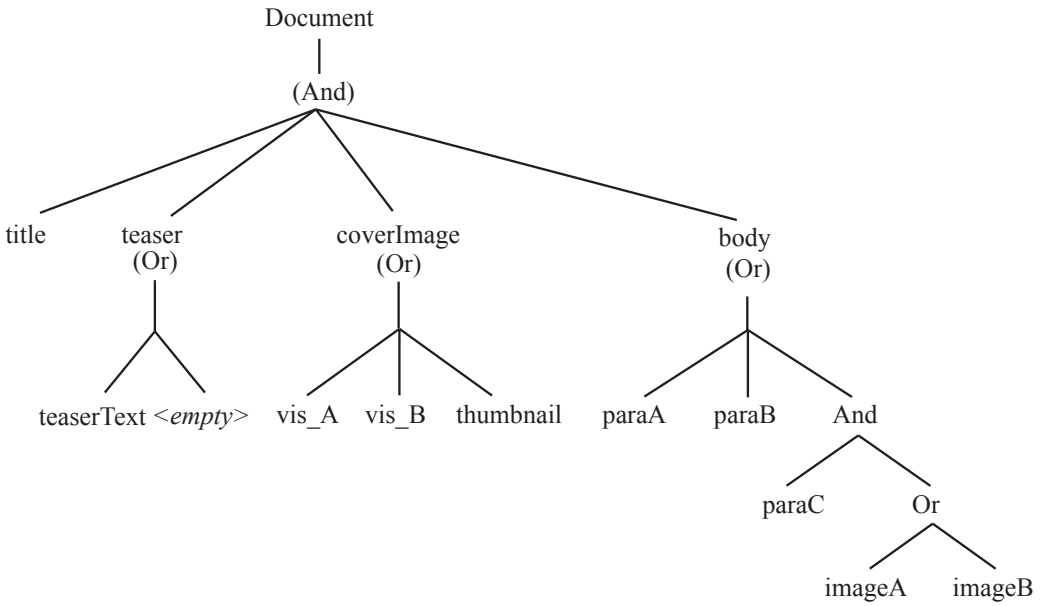


Figure 4.2.: Multi-representation document content tree.

Similarly, values of other attributes, such as `minFigureWidth` and `maxFigureWidth` (cf. line 38 and 41), and further attributes such as `minColWidth` and `maxColWidth` etc. can be specified inside `<Rep>` element using the same notation as in [SJL08]. These attributes serve as the means to select the appropriate content for various screen sizes and layouts. `<illustration>` element on line 15 of XML code gives a link to the annotated illustration file. The `interactive` attribute in `<illustration>` element denotes whether the illustration embedded into the document should support user interaction or not. The `userLevel` attribute on line 26, 29 and 34 indicates which content to select depending upon the knowledge level of the user.

4.3. Template Authoring

The potential of using design templates in digital media has already been discussed in the previous chapter. Within the scope of adaptive documents, templates provide the designers a convenient way to express the content and layout requirements which must be obeyed by the document framework while satisfying simultaneously the contextual needs imposed by viewer.

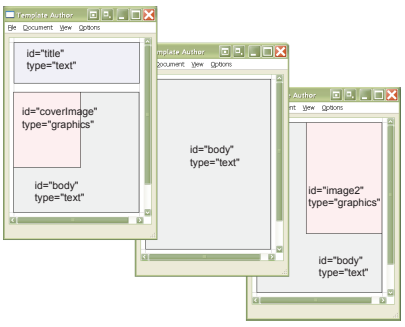
A graphical authoring tool is provided for the designers to create a set of layout designs (see figure 4.4) and specify their adaptivity to various screens. Designers draw layout elements,

```
<Document>
 <And>
  <text id="title"> ... </text>
  <Or id="teaser">
   <Rep minPageWidth="480">
    <text id="teaserText"> ... </text>
   </Rep>
   <Rep maxPageWidth="480" >
    <!-- empty -->
   </Rep>
  </Or>

  <Or id="coverImage">
   <Rep minPageWidth="480">
    <illustration src="vis_A.xml" interactive="yes"/>
   </Rep>
   <Rep maxPageWidth="480" >
    <illustration src="vis_B.xml" interactive="no"/>
   </Rep>
   <Rep maxPageWidth="240">
    <img src="thumbnail.jpg"/>
   </Rep>
  </Or>

  <Or id="body">
   <Rep userLevel="beginner">
    <text id="paraA "> ... </text>
   </Rep>
   <Rep userLevel="intermediate">
    <text id="paraB "> ... </text>
   </Rep>

   <And>
    <Rep userLevel="advanced">
     <text id="paraC"> ... </text>
    </Rep>
    <Or>
     <Rep minFigureWidth="300">
      <img src="imageA.jpg"/>
     </Rep>
     <Rep maxFigureWidth="300">
      <img src="imageB.jpg"/>
     </Rep>
    </Or>
   </And>
  </Or>
 </And>
</Document>
```

Figure 4.3.: XML representation for document content tree in figure 4.2.

and specify their size and position constraints by snapping the edges of containers to the typographic guide-lines. Each Container is tagged with the type of content that can be displayed inside, such as text, graphics or mixed. The process of authoring template is shown in the figure 4.5. Elements and spatial constraints on the elements can be added/removed by an author who can also modify the attributes of existing elements. The input constraints are resolved by a constraint solver which determines the new values for the position and size constraint variables of the layout. Finally, the template is drawn on screen using the adapted layout parameters.

(a) Template A in portrait format for desktop display.



(a) Template B in portrait format for PDA display.

Figure 4.4.: Template designs for two different screen sizes.

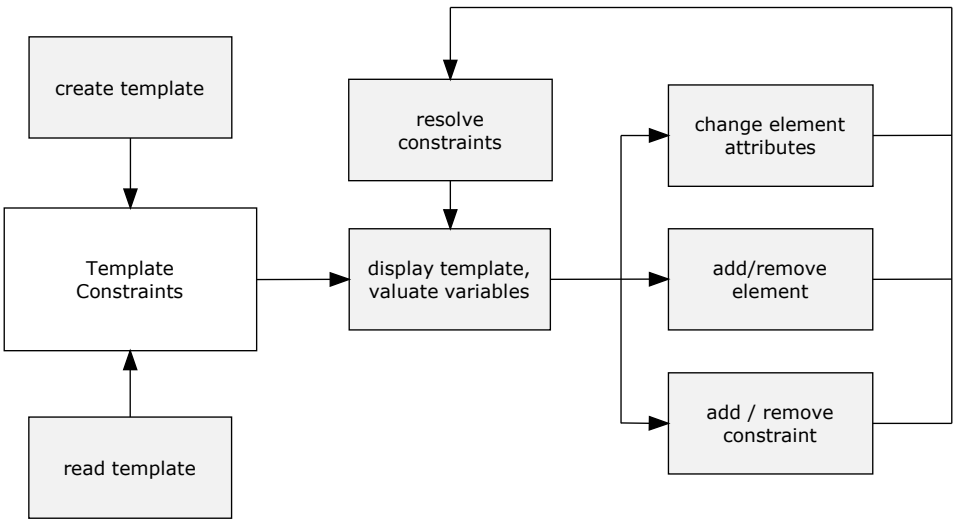


Figure 4.5.: Template authoring process (adapted from [TBR00]).

The design templates encode graphical constraints and are stored in the *template library*. With the templates and layout constraints stored therein, the layout engine can

automatically modify the layout on the fly to accommodate different versions of the content.

An illustration of design template is shown in the figure 4.6. The templates are encoded in the XML format. A `<Template>` element represents a design template.

```
<Template id="template1", minPageWidth=600, maxPageWidth=800>
 <Elements>
  .
  .
  .
 </Elements>
</Template>
```

The `minPageWidth` and `maxPageWidth` attributes specify a range of page sizes, known as *screen preconditions* to use the template—e.g., page width should be 600–800. An `<Elements>` element consists of a collection of `<element>` nodes in the template. An `<element>` represents a layout element in the template. Given below is an example of few template elements, in XML representation:

```
<Elements>
 <element id="title" type="text" contentSrc="title"/>
 <element id="teaser" type="text" contentSrc="teaser"/>
 <element id="coverImage" type="graphics" contentSrc="coverImage"/>
 <element id="body" column="col1,col2" type="text" contentSrc="body"/>
</Elements>
```

The `contentSrc` attribute maps a source content element to the template element. Using `contentSrc` attribute, the layout engine is informed which content to fill inside a template item. In this example, the four template elements fetch the data from their corresponding content elements given in figure 4.3. The content sources of template elements indicate the *content preconditions* for a template. By evaluating the *content preconditions* of a

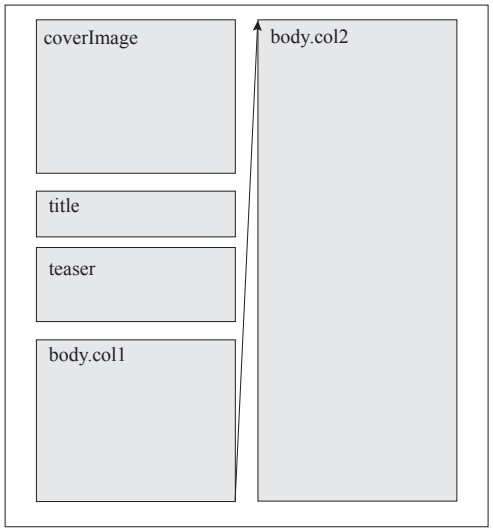


Figure 4.6.: A design template for document content given in figure 4.3.

template against the input document content, the layout engine determines whether the document content contains all required elements to appropriately fill the template.

Each of the layout elements of the template are assigned a position, and size (`width`,`height`). The edges of an element can be considered as `top`, `right`, `bottom`, and `left` attributes. Such attributes are also associated with global `page` element which is a rectangular element on screen to hold the document. Using the graphical user interface, the designer can impose a variety of spatial constraints on each `<element>` of the template:

- *absolute constraint:* an element is given an absolute position or size,
- *relative constraint:* the position and size of an element is defined in relation to another element. Two constraints are provided for convenience:
  - *adjacency constraint:* two elements are constrained to ensure that one stays above, below, to the left of, or to the right of the other,
  - *alignment constraint:* two or more elements are constraints to align vertically or horizontally to one another, and
- *group constraint:* two or more objects are grouped to stay together.

These constraints are translated to linear equality and inequality constraints which are represented in a notation as in CSVG (Constraint Scalable Vector Graphics) [BTM+01].

Using this notation scheme, a constraint is represented by `<constraint>` element. For example,

```
<constraint rule="image1.left>image2.right+50" strength="required"/>
<constraint rule="image1.top=image2.top" strength="required"/>
```

The `rule` attribute specifies the linear equality and inequality constraints. The first (adjacency) constraint in the aforementioned example declares that the image1 is to the right of image2 by more than 50 units. The second (alignment) constraint dictates that image1 and image2 are top-aligned.

In order to allow the designers to specify the preferences for constraints, the concept of *constraint hierarchy* is used. A constraint hierarchy consists of a set of constraints each labeled with its *strength*. The strength of a constraint is represented by the value of `strength` attribute inside `<constraint>` element. A strength value `required` denotes that the constraint must be satisfied [BBS01]. The other strength values, `strong`, `medium` and `weak` denote preferences from strongest to weakest. Stronger strength values are generally chosen to preserve the elementary structure of the layout, for example width and height of graphical objects in the page.

The layout constraints for the template design in the figure 4.6 can be expressed as:

```
<constraint rule="coverImage.top-page.top=constant" strength="strong"/>
<constraint rule="tile.top-coverImage.bottom=constant" strength="strong"/>
<constraint rule="teaser.top-title.bottom=constant" strength="strong"/>
<constraint rule="body.col1.top-teaser.bottom=constant" strength="strong"/>
<constraint rule="body.col2.top-page.top=constant" strength="strong"/>

<constraint rule="coverImage.left-page.left=constant" strength="strong"/>
<constraint rule="tile.left-page.left=constant" strength="strong"/>
<constraint rule="teaser.left=tile.left" strength="strong"/>
<constraint rule="body.col1.left=tile.left" strength="strong"/>
<constraint rule="body.col2.left-body.col1.right=constant" strength="strong"/>
```

The above XML code shows only position constraints of template elements. Layout constraints on the size of template elements can be expressed in the similar manner:

```
<constraint rule="body1.col1.width-page.width/2=constant" strength="strong"/>
<constraint rule="body1.col2.width=body.col1.width" strength="strong"/>

<constraint rule="coverImage.width=body.col1.width" strength="strong"/>
<constraint rule="tile.width=body.col1.width" strength="strong"/>
<constraint rule="teaser.width=body.col1.width" strength="strong"/>

<constraint rule="body.col2.height-page.height=constant" strength="strong"/>
<constraint rule="body.col1.bottom=body.col2.bottom" strength="strong"/>
```

The set of given constraints in the template are fed into a constraint solver for resolving. Adaptive document layout approach presented in this thesis uses Cassowary Constraint Solver [BBS01] which allows to establish a hierarchy of constraints, such as `strong`, `medium` or `weak`. Moreover, the Cassowary is an efficient incremental solver. Constraints can be added to or removed from Cassowary incrementally, which resolves them by taking into account the existing solution instead of recomputing the solution from the scratch.

4.4. Task Driven Authoring

Up to now, this chapter has only discussed the aspects of authoring alternate content, designing templates, and specifying rules for the content and template adaptation. Nevertheless, the *information needs* of the user in a given context are not discussed so far. One of the goals of the dynamic media is to provide users with the information that is relevant to their *tasks*. Therefore, it may be necessary to prioritize content elements in the presentation and emphasize the information according to their relevance. This scenario is common in repair and maintenance work where the users follow step-by-step procedures to carry out specific tasks. A *task-driven* presentation of content maintains that the information needs and presentation style are determined by the task at hand.

Their capability to change continuously over time make adaptive documents a primary choice for the task-driven adaptation of content and presentation.

The tasks can be stored in a so-called *task model*. A *task statement* within a task model refers to a set of activities that are performed to achieve a particular goal [DOU93]. For example, the activity of adjusting the view of a presentation may correspond to the task of making an object visible. A task model helps identifying the goals of the system and breaks down the functionality of system into individual tasks. A widely used data structure in this regard is ConcurTaskTree (CTT) task model that forms a hierarchy of compound tasks with subtasks [PMM97]. Subtasks have conditional and/or temporal relationships between them, and represent the step-by-step nature of individual working procedures.

Task models are commonly used in user interface design to analyze activities carried by humans as well as those performed by machines to achieve a goal [MPS02], and for automatically creating a user interface. Another area where task models have found their application is the field of interactive visualization where CTT task models are used to specify the tasks that can be performed in the visualization [WL90]. In "e-Manual" application [FRSF06], ConcurTaskTree (CTT) task model is extended for the task-driven presentation of different types of visual content on handheld devices. Here, a task model enriches the content by associating graphical content and defining visual representation associated with each task.

Presentation of relevant information to the user is greatly backed by task model, since the task model removes the information out of context. This is particularly useful for small screen sizes where the information density of a given presentation has to be carefully managed for the visualization to remain effective. In adaptive document framework, CTT task model can be extended to enable task-driven generation of adaptive documents. Editors or content authors can specify different priorities per task for content elements that will affect how the content is chosen and laid out by the layout engine. Integrating the layout engine with the task model would afford a task-driven generation of initial document views. Subsequent interaction by the user still allows for dynamic layout changes, including updates of content priorities and even automatic selection of another template based on importance value changes derived from user interaction. In this way, the task model in the adaptive document framework would provide an authoring interface between the content and presentation.

An example of a task model for adaptive documents is given in the figure 4.7. The `<taskModel>` element represents a task model whose task is to generate a graphical presentation for the subject matter `"How Human Heart Works?"`. A `<task>` element represents one task within the task model. The task description is `"Introduction to Human Heart"`. The `style` attribute gives the preference order of templates that can be chosen by the layout engine to present the contents. A `<task>` element consists of a set of `<element>` nodes to provide links to content. The `id` and `contentSrc` attributes inside a `<element>` create a link between source content element and container inside the template. Therefore, any differences of element tag names between the content model and templates can be fixed using task models. This allows the document framework to maintain the separation between the content and style. The `value` attribute `<element>` can be used to provide actual content itself (line 4).

Interactive visualizations can apply graphical emphasis techniques to highlight objects needed within each task. A task model provides the visualizations which objects are relevant to the task. An `<illustration>` element on line 6 represents an interactive illustration. Each of the graphical objects in the illustration are addressed by an `<item>` element. The relevance of an `<item>` in the given task is indicated through its `importance` attribute. For example, importance value 1.0 of item on line 8 denotes its higher relevance to the subject matter when compared to item on line 9 which has importance value 0.6. An importance value 0.0 on line 10 indicates no relevance. A higher `importance` value instructs the layout engine to highlight the item in the illustration and present it in detail with annotations.

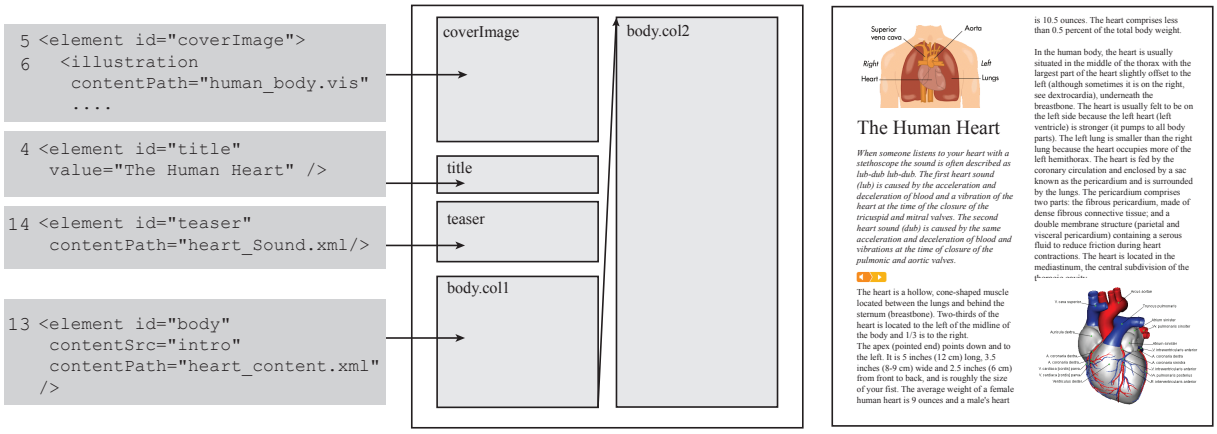
At runtime, the task model is processed by the layout engine to generate document views. For the task at hand, content is selected from `<element>` nodes and candidate set of templates is established using `style` attribute. For interactive visualizations, appropriate viewpoint for relevant objects is selected with attached annotations. All this, together with the viewing parameters, leads to the process of computing final output by the document layout engine. Figure 4.8 gives a working example of the task model listed in the figure 4.7. Based on the task model, layout engine collects content from different sources and pours it in the template (originally given in the figure 4.6). In the final output, the relevance parameters (cf. line 7–10 in figure 4.7) for graphical objects are reflected in the (top-left) interactive illustration.

```
<Taskmodel id="heartTaskModel" name="How Human Heart Works?">
 <Task id="task1" name="The Human Heart"
       style="templateA,templateB,templateC">
  <element id="title" value="The Human Heart" />
  <element id="coverImage">
   <illustration contentPath="human_body.vis" annotate="yes" interactive="yes">
    <item id="feature_body" importance="0.4" />
    <item id="feature_heart" importance="1.0" />
    <item id="feature_lungs" importance="0.6" />
    <item id="feature_liver" importance="0.0" />
   </illustration>
  </element>
  <element id="body" contentSrc="intro" contentPath="heart_content.xml" />
  <element id="teaser" contentPath="heart_Sound.xml/>

 </Task>

 <Task id="task2" name="Blood Flow in Heart"
       style="templateX,templateB,templateC">
  ...
 </Task>
 ...
</Taskmodel>
```

Figure 4.7.: A task model for adaptive document.

4.5. Discussion

This chapter described the authoring steps in the domain of adaptive documents. First step is the authoring of the content. Since automated content authoring is a difficult problem because of the involvement of various factors, such as user modeling, information analysis and retrieval, etc. the approach of the thesis rather relies on the manual authoring of content. Using a multi-representation content scheme adopted in this work, the authors can create alternate versions of content which allows the layout engine to select the best version of the content to match the various viewing and contextual requirements. Second step of authoring process is the template designing. Designers are provided with a template authoring tool to construct the layout design and specify the spatial constraints that are necessary to maintain the the overall layout. Third step of the authoring is the task-driven mechanism. Traditional CTT task model is extended to facilitate the document adaptation in accordance to the task at hand. This novel approach allows the adaptive document framework to create different views of underlying content and emphasize the graphical content in the presentation to reflect their relevance in the given scenario. The result of the authoring process serves as a base for the layout engine, explained in the next chapter,



(a) Task model (b) Template (c) Final Output

Figure 4.8.: A document view based on task model. *a:* The task model elements from figure 4.7. *b:* The template from figure 4.6. *c:* The final document.

which collects viewing parameters of the display device at run-time to generate a formatted view of the document.

Chapter 5

The Layout Engine I: Adaptive Document Layout

This chapter presents a novel approach to create adaptive document layouts from templates, content and task models. It discusses issues when the template-based approach alone is not sufficient in the layout process. An integrated solution is developed to generate layouts which takes into account both the content requirements, semantic relationships within the content and other layout requirements. Using the techniques proposed in the chapter, optimal layouts can be created for rich illustrative material even when the templates are partially designed. The integrated approach allows the document to retain the underlying design of the template when making necessary adjustments in the layout.

Section 5.1 gives an overview of the proposed approach. The first step in the layout process is content and template selection based on the contextual requirements. The details of this step are given in section 5.2. Section 5.3 is the heart of this chapter which shows how various content and layout requirements in the document can be modeled to a set of physical forces which are used to adapt the content and layout within the page. Moreover, a unified solution is given to combine the results of traditional constraint-based techniques with the newly proposed force-directed document layout. The results indicate that the new strategy expands the capabilities of document layout engine to deal with variable content at different screen sizes. In section 5.4, we discuss an efficient pagination scheme that simultaneously layouts both the text and figures into pages. Section 5.5 presents

results and gives reflections about layout parameter settings. Section 5.6 concludes the chapter with the discussion.

5.1. General Approach

In digital media, which deals with dynamic information and continuously change of readers' requirements, the designers have to create design solutions that can evolve over time (cf. section 3.1). The key to this problem is to create a right mix of custom-made templates and automated adaptation. This would allow the system to make necessary adjustments in template to satisfy the presentation requirements while leaving the underlying design of the template intact.

Design properties, such as placement of content on the page, aesthetics etc. can be defined through constraints in the templates. At design-time, a designer can create a range of templates for various screen sizes, content structure and layout styles. But instead of giving absolute values for size and positions of layout elements, a designer can specify a range of values that the system is constrained to choose when adapting the design. These design constraints are combined with a scoring mechanism to evaluate various design alternatives at run-time. At run-time, the display device imposes additional constraints on the design. All of this information is analyzed by document layout engine to decides which template to use. The layout engine modifies the template by choosing an acceptable value for each layout parameter which is constrained by the designer, the viewing device, and the content itself.

This concept is illustrated in figure 5.1. Client's display device is attached to the layout engine. The layout engine has an access to multiple alternatives of content and a set of design templates provided in the template library. During each production cycle of a document, the job of layout engine is to select a particular version of content and suitable template and to tailor the document to the display device.

Figure 5.2 shows the steps followed by the document layout engine. First part of document layout engine, layout engine I (adaptive document layout) or simply referred as layout engine in this chapter, determines the placement of text and figures in the document. It computes a design solution that satisfies the constraints imposed by designer and the display device. Finally, with a given content and design template, the goal of layout engine is to determine the number of pages and assign a position as well as size to each element

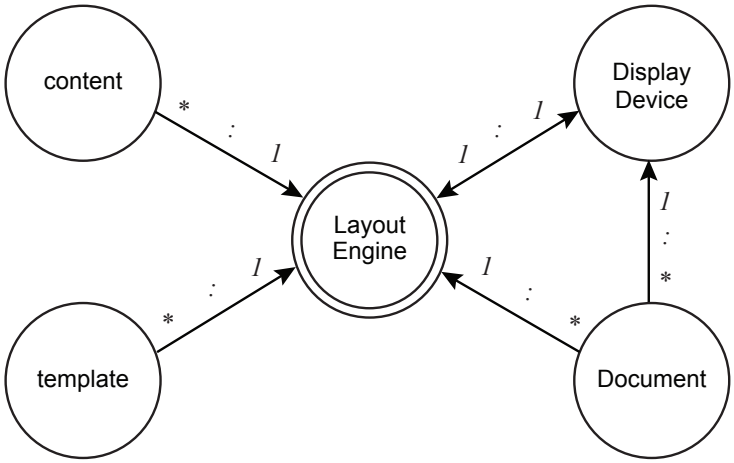


Figure 5.1.: Input and output to layout engine in adaptive document framework. In this diagram, $1:1$ denotes one-to-one relationship, $1:*$ denotes one-to-many and $*:1$ denotes many-to-one relationship.

on the page within the viewing and design constraints. The layout engine accomplishes this task in three steps, *content and template selection*, *page layout* and *pagination*.

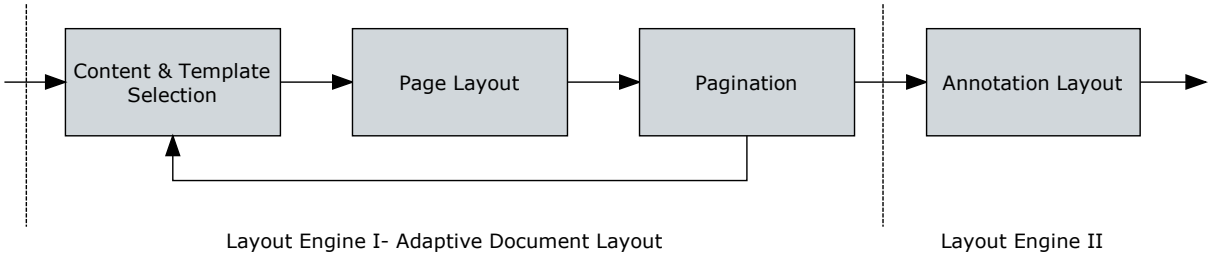


Figure 5.2.: Stages of document layout engine. The line from pagination to content & template selection indicates that the procedure is repeated to find optimal templates for subsequent pages of the document and lay out content.

Content and template selection stage evaluates the source content against the design templates given in the template library. It chooses a suitable page template for the optimal layout of given content on the display device.

During the *page layout* stage, the layout engine determines the final look of each page by resolving design and viewing constraints. The page layout is computed using two methods. The first method uses traditional constraint-based optimization to adapt the layout by resolving constraints given in the template. The second method, a force-directed algorithm, is applied to modify the parts of constraint-based layout. Force-directed approach is integrated with constraint solver so that the design constraints are not violated in the force-driven layout.

Once the layout engine-I has computed the size and position of regions in the design template, it flows the content into the layout regions according the stylesheet which contains formatting information such as font type, size, and other properties. Depending on how well the content fits into template, the *pagination* stage determines how the content would be allocated to a set of pages.

For illustrated material, the layout process also involves the challenging task of adapting illustrations within the document. The second part of layout engine, layout engine-II (annotation layout) which is elaborated in chapter 6, does exactly that. It adapts the annotation layout of illustrations within the page to match the contextual requirements.

5.2. Content and Template Selection

The layout process begins with the selection of candidate templates from the template library and ends up formatting the document with an appropriate version of content in the selected template. Content and template selection mechanism is illustrated in the figure 5.3 and is adapted from [JLS+03b]. Template selection refers to the process of identifying how suitable a template is for a particular screen size. At design-time, a designer specifies a range of screen dimensions called screen preconditions and content preconditions to use the template (cf. section 4.3).

In order to be able of filling the content appropriately in the template, the document content must fulfill the content preconditions of the template. Using the screen preconditions and content preconditions, the system is able to choose a suitable template for the current screen size and content. The procedure is elaborated in the following.

Typically, a template design allows a good layout adaptation only for a range of page sizes and aspect ratios indicated as screen preconditions by a designer. At run-time, the layout engine tests whether the current screen dimension fulfills the screen preconditions for the given template. If so, the template is added to the *candidate set*, otherwise it is discarded.

The candidate set of templates is further refined in the *content preconditions* step. Content preconditions are stored in the template by the designer and can also be specified in a task model (cf. chapter 4). The layout engine matches the `contentSrc` attributes of template element to the `id` attributes of content items (cf. section 4.3). The templates matching to the content structure are retained in the candidate set, otherwise they are discarded.

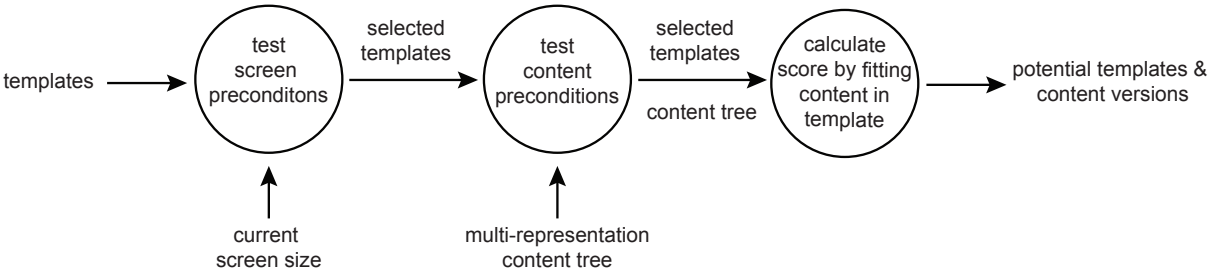


Figure 5.3.: Content and template selection process.

During the page layout step, the layout engine computes template scores in order to select the best templates from the candidate list. For each valid template in the candidate set, it adapts the template to the screen size and flows the content into template regions to find how good the content fits in the template. For example, if the dimensions of a figure are considerably different to that of figure container in the template, this can yield a poor score. These scores are used by the paginator to choose the optimal set of templates to layout the whole document. At this stage, the layout engine can also choose best version of content to layout the potential templates. For example, if one version of a figure content has its attributes set to `minFigureWidth=200 maxFigureWidth=400`, and the second version of the same element has its attributes set to `minFigureWidth=600 maxFigureWidth=1000`. In this case, the second version of content would get a higher score for a template element with `width=800` and thus preferred in the layout. This way, goodness score for each of alternate content is measured and combined to determine the overall goodness score. The content version with the best scores is chosen for the layout consideration.

Over the course of user interaction with the document, the layout engine follows the same procedure to opt for a different template on the basis of user preference, screen resizing or content update.

5.3. Page Layout

This stage of layout engine is responsible for adapting the template design to fit the content on page. For each candidate template, the page layout is determined by resolving the constraints for each element in the template and then flowing content into elements. While the past research in document layout has focused on constraint-based layouts, in this work we introduce force-directed methods to generate adaptive layouts for documents. An integrated solution is proposed, which combines the constraint-based layout with

force-directed layout to extend the flexibility in layouts, which be will discussed later. The layout process is carried out at two levels. The first levels uses constraint-based optimization to compute positions and sizes of constrained elements. At second level, force-directed technique is used to modify the constraint-based solution, or produce a new solution, either in parts or whole. During the force-directed phase, the layout adjustments are made only when they don't conflict with the constraint-based layout.

5.3.1. Constraint-Based Layout

Section 4.3 presented a template authoring tool that the designer can use to design templates. The design templates essentially represent spatial constraints specified by the designer at design-time, such as `title.left=50` or `fig.width=column.width`. At run-time, the display device imposes further constraints on the layout, such as `page.width=800`. Considering both the design-time and run-time constraints, the document layout problem can be modeled as a constraint optimization problem (cf. subsubsection 2.3.1.2). The constraint-based optimization seeks to resolve the constraints and assigns each *layout variable* a particular value from its value domain. This refers to using constraint solving methods to find the values of size and position variables of each layout element while satisfying constraints.

In this work, constraint-based optimization is used to satisfy the spatial constraints given in the template. The final layout of document is determined by the combined efforts of constraint-based optimization and force-directed algorithms (cf. next subsection). In order to resolve a system of constraints, the proposed layout engine uses Cassowary Constraint Solver [BBS01] which adapts simplex algorithm, a popular algorithm for solving linear programming problem [Nas00]. Cassowary solver was chosen because it allows to establish a hierarchy of constraints, such as `strong`, `medium` or `weak` (cf. section 4.3). Moreover, Cassowary is an incremental solver, it can modify the existing state by adding new constraints one at a time, and thus avoid recomputing solution every time from scratch. This feature increases Cassowary's processing speed and makes it a suitable choice for applications where speed is of utter importance.

Cassowary provides an abstract `ClLinearConstraint` class for specifying linear constraints. It has two subclasses. An instance of `ClLinearEquation` represents the linear equality constraint $expression = 0$, where $expression$ is a linear expression in the form of $a_1x_1 + ...a_nx_n$. An instance of `ClLinearInequality` represents a linear inequality

constraints *expression* ≥ 0. For example a constraint such as $2 * x + 5 < y$ is written as `ClLinearInequality cn(2*x+5,cnLEQ,y);`

To use the CASSOWARY solver, spatial constraints are converted to linear constraints. A parser is developed which reads the spatial constraints given in the template (cf. section 4.3) and translates them to the format of linear equality and inequality constraints understood by CASSOWARY solver. Assume that a design template contains four rectangular boxes (B1, B2, B3 and Fig) and a set of constraints (cf. figure 5.4). Given below is an example of verbal descriptions of some constraints, their XML representation in the template file, and CASSOWARY instructions in C++:

- Constraint: B1 has the same width as page
  XML: `<constraint rule="B1.width=page.width" strength="strong"/>`
  C++: `ClLinearEquation cn(B1Width,pageWidth,ClsStrong());`

- Constraint: Fig is square
  XML: `<constraint rule="Fig.width=Fig.height" strength="strong"/>`
  C++: `ClLinearEquation cn(FigWidth,FigHeight,ClsStrong());`

- Constraint: Fig is placed in horizontal center of page
  XML: `<constraint rule="Fig.center=page.width/2" strength="strong"/>`
  C++: `ClLinearEquation cn(FigCenter,pageWidth/2,ClsStrong());`

- Constraints: Fig, B2, and B3 are below B1
  XML: `<constraint rule="Fig.top>B1.bottom" strength="strong"/>` and likewise so on so forth
  C++: `ClLinearInequality cn(FigTop,cnGEQ,B1Bottom,ClsStrong());` etc.

- Constraints: Fig, B2, and B3 are top aligned
  XML: `<constraint rule="Fig.top=B2.top" strength="strong"/>` etc.
  C++: `ClLinearEquation cn(FigTop,B2Top,ClsStrong());` etc.

- Constraint: B3 is to the right of B2
  XML: `<constraint rule="B2.right<B2.left" strength="strong"/>`
  C++ code: `ClLinearInequality cn(B2Right,cnLEQ,B2Left,ClsStrong());`

- Constraints: Fig, B2 and B3 are each half the width of page
  XML: `<constraint rule="Fig.width=page.width/2" strength="strong"/>` etc.
  C++: `ClLinearEquation cn(FigWidth,pageWidth/2,ClsStrong());` etc.

The CASSOWARY solver is represented by an instance of `ClSimplexSolver`. A constraint is added to the solver by making a call to `addConstraint(ClLinearConstraint cn)` method of `ClSimplexSolver`. The `resolve()` method resolves the system of constraints and sets the constraint variables to new values. Figure 5.4 shows the example layout after as a result of constraint-based optimization. To achieve the layout, first the layout engine computes the constraint-based layout according to the current page size without content. Then it flows the content into regions. Figures are scaled to match fit inside the figure containers and text is flowed inside elements one by one, as described in the next section. The height of elements is auto-adjusted according to the amount of text. After flowing content in each element, new values of the regions are suggested to the ClSimplexSolver solver. ClSimplexSolver resolves the constraints and returns updated values for layout elements. In this example, the layout engine automatically adjusts the height of block B1 to fit the content. The adjusted height value of the B1 is then used to update the values of other constraint variables affected by B1.

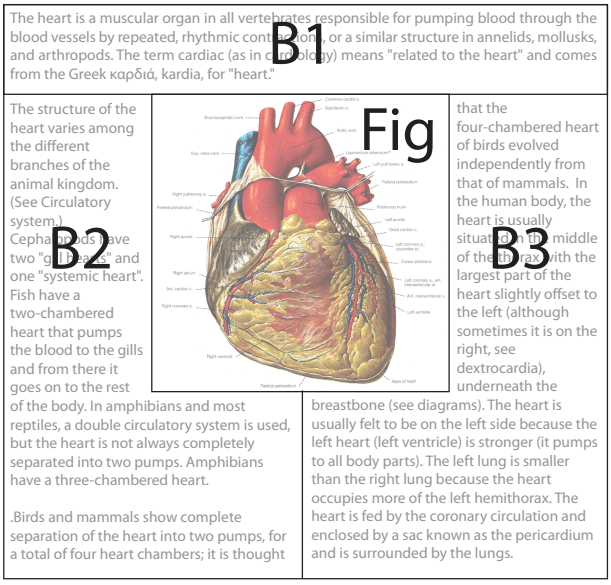


Figure 5.4.: Constraint-based layout based on simplex algorithm.

At run-time, when the document is manipulated, for example when the user resizes the screen, the values of page width and height constraint variables in `ClSimplexSolver` solver are updated. The solver resolves the constraints according to the new input values, and return the page layout. The layout engine then chooses an appropriate version of content and flows the content inside elements. The solver is invoked every time there is a change in the element size due to content filling. Graphics can be placed either in the elements reserved for them or as in-line elements pasted directly into the flow of text. During the

course of interaction, if the current screen size is not accepted by the template, another template is chosen, as explained in previous section.

Even though constraint-based optimization methods are widely used in layout applications, there still remain few challenges. If the number of constraints is fairly large and contains contradictions among constraints, it can make system of constraint unsolvable. Constraint hierarchies may handle such inconsistencies by dropping the less significant constraint involved in the conflict, but the problems still occur when more than one constraints with equal strengths are involved in the conflict, in which case it requires further analysis to decide which constraint(s) to drop.

Another issue with constraints is because of the fact, that many of the key layout requirements (such as edge alignment and adjacency) can be represented using linear constraints very precisely and in a convenient fashion. This makes constraint-based optimization a suitable choice for grid-based layout [JLS+03b]. The underlying typographic grid is first laid out on the page onto which the edges of template items are snapped to. In this way grid controls the size and positions of template items placed on top of grid. Even without specifying an underlying grid, the edges of rectangular elements denote an auxiliary grid lines upon which are used as reference to slide the edges of other elements. Therefore, constraint-based are mainly used for grid-based design.

A constraint-based is only able to handle spatial constraints among layout elements, which is affordable when working with the design. But, within content structures, only abstract constraints are specified, such as `FIG1 is IMPORTANT` and `TEXT1 REFERENCES FIG1` (cf. subsubsection 2.3.1.2). The content authors may wish that the semantic relationships they have encoded through abstract constraint are also reflected in the final layout. A constraint-based layout system cannot deal with abstract constraints, because abstract constraint are not in the form of linear constraints. Thus, abstract constraints have to be first translated to concrete spatial constraints before they are passed to the constraint solver. But, an abstract constraint may be a vague description—i.e., place related items together—and translating it to a linear equation representing a spatial constraint may not be precise. As a consequence, the constraint-based layout may not be able to truly reflect the semantic relationships among the content. In the next subsection, a novel force-directed layout technique is presented that overcomes such problems. It translates abstract constraints to physical force and integrates them to spatial constraints to find an optimal layout.

5.3.2. Force-Directed Layout

The reason that a constraint based layout is characterized by spatial constraints given at design-time also makes it difficult for the layout engine to format the document if the content structure given at run-time is drastically different from what was originally conceived by the designer. For example, assume a scenario where the content is not provided for a graphical element in the template. Using the template would leave an empty region in the final output. A grid-based constraint optimization placement, in case, may exhibit an inefficient use of page area. A situation arises in rich illustrative media, when the size and number of illustrations on the page are unknown at design-time. How to design a layout for such content? Further assume that only abstract constraints are given which only describe the importance of elements in the topic and correlation among elements. How to compute a layout if the only given requirement is that there be no element overlaps in the layout and that related items are placed close to one another? A design merely based on such requirements enlarges the solution space significantly and poses many challenges to the restricted template-based constraint layouts because the design qualities cannot not be precisely described in a convenient fashion.

As pointed out in subsubsection 2.3.1.3, one way to implement constraints is by using force-directed approach. In past, force-directed methods have been widely used in drawing graphs [DETT99]. But, this thesis takes a novel approach and models the document layout problem into force-directed layout problem. Thus, the task of finding a good placement of content items on the page primarily on the basis of abstract constraint is modeled by forces that cause alterations in the size and positions of element in the layout.

Force-directed algorithms are used in this work to achieve non-grid layouts of illustrated material with or without using templates. Non-grid layouts are widely used in rich illustrative material such as visual dictionaries, anatomy books and assembly instruction manuals etc. Unlike a grid layout which restrict the graphic elements to be of integral proportion of the grid cells, a non-grid layout finds a placement for a set of arbitrary shaped elements in a organic way. Non-grid layouts look particularly better compared to grid-layouts when the figures are non-rectangular or have transparent background color. In this scenario, a grid-layout does not provide enough tension in the layout, as a result non-rectangular graphics may appear floating within a blob space. Therefore, in order to integrate a set of arbitrary shaped annotated illustrations, possibly correlated to each other, in a seamless fashion, a non-grid layout becomes natural choice.

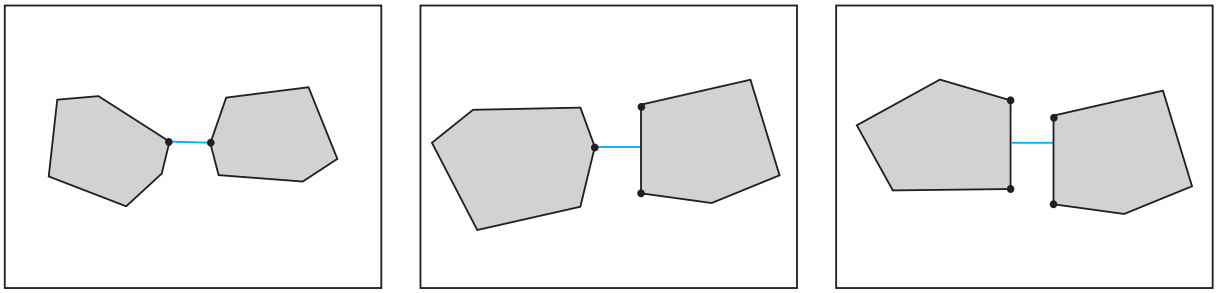
Force-directed methods were originally developed for drawing graphs, and haven't been explored in the field of document layout. Taking a novel approach, this thesis conceives using force-directed graph drawing methods to generate adaptive documents layouts. Let us formulate document layout as *graph-layout* problem. A document is represented by graph G. Vertex nodes V of graph denote convex polygonal shaped elements in the layout, typographic guide-lines, and area boundary lines. Edges V of graph model relationships between nodes. For example an edge can model dependency between two content items that are correlated to each other. An edge between a typographic grid line and a content item can be used to indicate the alignment of item to the grid line. Similarly an item can be anchor to a location using an edge.

Using a document graph model, force-directed drawing algorithms can be used to compute the placement of elements in the document. But, most of the graph drawing algorithms only deal with graphs with zero-sized nodes, though there are some extensions to produce graph layouts for nonzero sized nodes [HK02].

But, there are some basic differences between document layout and graph layouts. First, the dependency edges of the document graph are only used to compute placement, and are not drawn explicitly in the final document as opposed to graphs where they are drawn. The semantic relationship among items in the document layout is made clear mainly through adjacency and alignment constraints. Second, edge-edge crossing and edge-vertex crossing is not significant in documents, but for graphs it is. Third, the nodes in the document graph can hold variable content, which requires that nodes can be resized arbitrarily and moved without overlapping. Also, if the templates are given, we need a way to combine the traditional constraints with forces. Thus, the traditional force-directed graph drawing algorithms need to be adapted to use in document layout. The rest of the section describes the novel approach which enables us to deal with the document layout problem using a system of forces and integrate them with traditional constraints.

5.3.2.1. Force-Directed Placement

Force-directed methods, described in this subsubsection, are aimed at finding an optimal placement for floating elements (i.e., figures, text boxes). The floating elements can be moved anywhere within the layout boundary. The physical forces are derived from the content or the template.



(a) Polygonal vertex-vertex distance (b) Polygonal vertex-edge distance (c) Polygonal Edge-edge distance

Figure 5.5.: Minimum distance between two convex polygons [Pir99].

In force-directed layout, each element is modeled as a node and the semantics between elements are modeled as edges or physical forces exerted upon them. A set of abstract and spatial constraints are turned into forces which can be applied to displace layout elements. Original graph drawing methods were enhanced at many levels to deal with various non-zero sized shapes, such rectangles, arbitrary convex polygons, as well as line elements such as typographic guidelines, or boundary lines in the layout.

To achieve accurate results for arbitrary shaped elements, minimum distance between two elements is determined by anti-podal pair between the convex-shaped polygons [Pir99] (see figure 5.5). It is also used to compute distance between a polygon and a line element.

Four types of forces are established:

1) Local Spring Attractive Force: A local spring attractive force is established between edge-connected nodes. Original attractive force function F_a [FR91] is chosen for this purpose.

$$F_a(u, v) = d(u, v)^2/k_s$$

Here d is the distance between u and v, whereas k_s is spring constant which models the natural length of spring.

Overlaps may occur as a result of attractive force displacement. The overlaps are resolved by other forces (explained later). The attractive force aims to satisfy following abstract and spatial constraints.

- *Correlation:* Referenced element should be placed close to the element which refers to it. A spring between two related element u and v is bound to keep them close. u

and v are considered related if u cites to v, and when two elements have the same parent in the content tree.

- *Preferred Location:* An element should be placed to its location specified in the template. If an absolute target location was specified by the designer, the element is statically anchored to it and no springs are required. Otherwise, a spring is attached between the current and target placement of an element, if given in the template.

- *Alignment:* To enforce the alignment, a spring is attached between each floating element and the alignment edge. The spring force snaps the element to the typographic grid or auxiliary guide-line.

- *Temporal Coherence:* To avoid jittery movements in the layout, a floating element should maintain its position on the page over the time. For a frame-coherent layout, an element can be spring-attached to its location in the previous layout.

2) Global Spring Repulsive Force: A global Spring repulsive force is calculated between every pair of nodes (edge-connected and unconnected) in the layout to avoid overlaps. It is also computed between nodes and boundary walls to keep the nodes within the page. An illustration of attractive and repulsive forces in the document layout is shown in figure 5.6. The repulsive force function is defined [FR91] as:

$$F_r(u, v) = -k_s^2/d(u, v)$$

3) Global Gravitational Attractive Force: In an unconnected graph, a global spring repulsive force pushes the nodes farther apart because there are no attractive springs attached. To counter this, a global gravitational attractive force, analogically similar to the global repulsive force, is applied in order to keep the nodes from drifting far apart. Original gravitational force function was developed for drawing zero-sized nodes [FLM95], therefore, we need to adapt it non-zero sized nodes.

In the adapted model, the gravitational force drags the convex-shaped nodes towards the barycenter ζ of the graph, which is given by $\zeta = \frac{\sum_{i \in V} C_i A_i}{\sum_{i \in V} A_i}$. Here, C_i is the centroid of node i and A_i is the area of i. The denominator of gravitational force is defined by

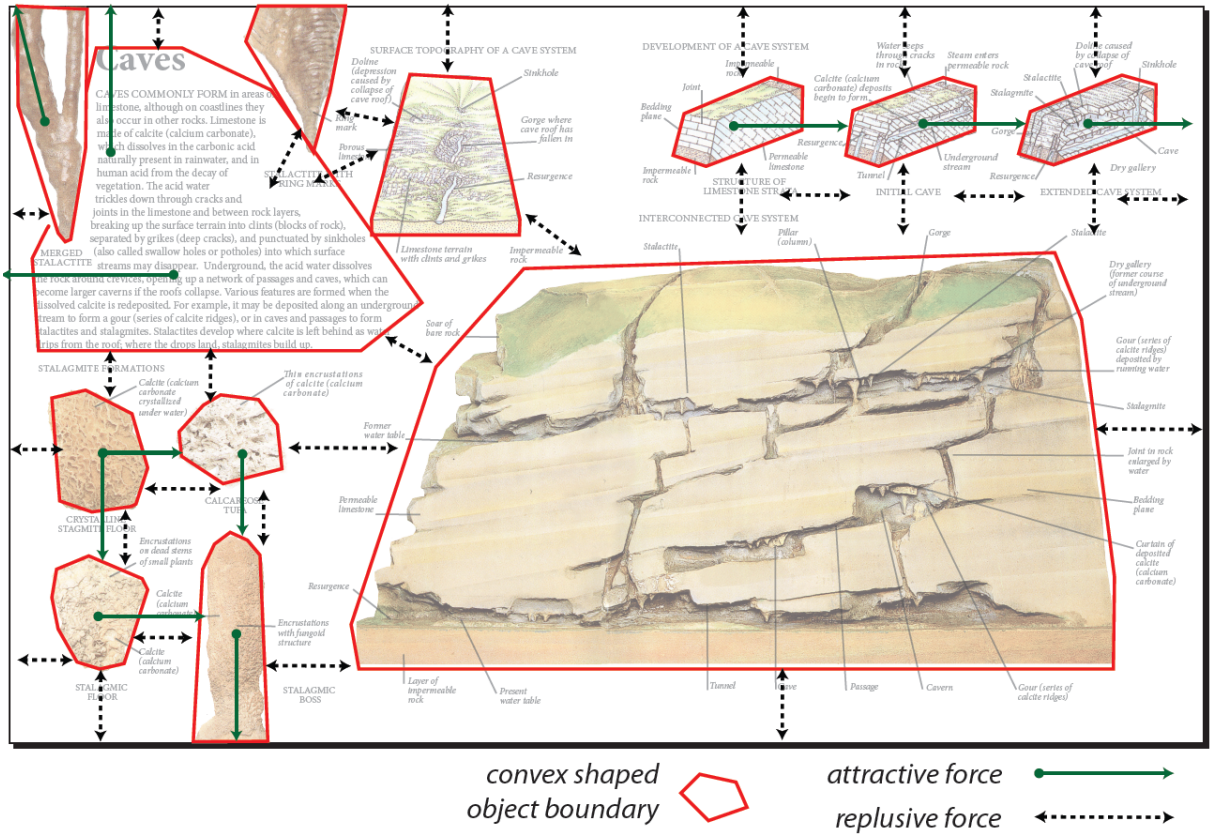


Figure 5.6.: Attractive and repulsive acting on elements.

$\Phi(v) = 1 + \frac{deg(v)}{2}$, which is a stronger function for higher-degree nodes. The gravitational force function itself is defined as:

$$F_g(v) = k_g \, \Phi(v) \, (\zeta - C_v)$$

where k_g is the gravitational constant which determines how strongly the node is pulled from its centroid C_u towards the barycenter ζ of the layout.

4) Spring Magnetic Field: Aforementioned attractive and repulsive forces do not take the direction of edges into account. In directed graphs, each edge has a uniform direction too. Sometimes the graph layout requires that the edges of the graph are conformed to specified orientations which is accomplished by the spring magnetic field [SM95]. In the force-directed document layout, we can use magnetic field to fulfill the adjacency constraint between two nodes and align them. Assume two adjacency constraints, u left-to v. This constraint requires the elements u and v to appear in a specific horizontal order. $u \rightarrow v$ constraint establishes a orientation of the magnetic field m. During force-directed layout

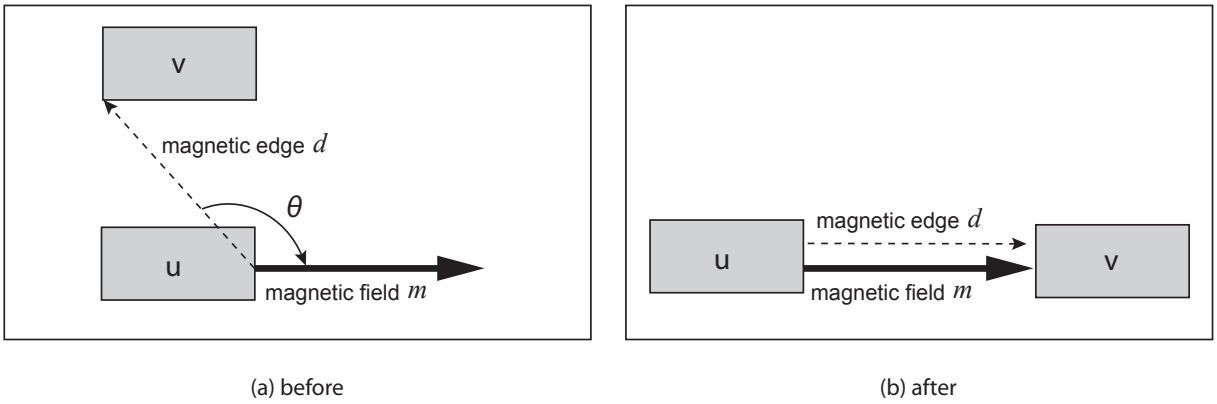


Figure 5.7.: Edge orientation conforming using spring magnetic field.

phase, magnetic force is calculated from the angle formed by the magnetic edge d between u and v, and the magnetic field m (see figure 5.7). Spring magnetic force is defined by:

$$F_m(u, v) = k_m \, d(u, v)^{\alpha} \theta^{\beta}$$

where k_m, α, $\beta > 0$ are constants for tuning the layout, and θ is the angle (in radian) from the the orientation of m to the orientation of magnetic edge.

Figure 5.8 shows the screenshot where adjacency constraints of great importance in order to present the illustration figures in correct reading order. Following adjacency constraints are resolves by the spring magnetic field—$Fig1 \rightarrow Fig2 \rightarrow Fig3 \rightarrow Fig4$—in order to present the illustration figures in correct reading order. Following adjacency constraints are resolves by the spring magnetic field: $Fig1 \rightarrow Fig2 \rightarrow Fig3 \rightarrow Fig4$.

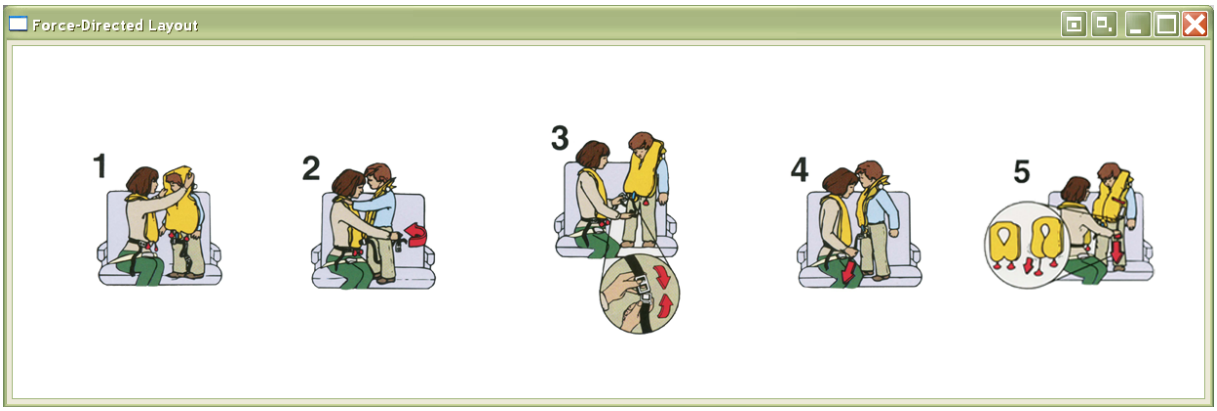


Figure 5.8.: The adjacency constraints are resolved by the spring magnetic field to maintain a correct reading order in figures.

In order to obtain the equilibrium, nodes are iteratively moved over the time according to the net force $F_v(i)$, which is equal to the sum of attractive, repulsive, gravitational and magnetic forces. After computing $F_v(i)$ for all nodes, each node is moved in the direction of force vector by the factor of its magnitude. By iteratively computing the displacement of all nodes and updating their positions, the system approaches the equilibrium state. Over the course of iterations, the magnitude of node displacement is controlled by annealing temperature T that is initially set to 1. T cools down by each iterations. Algorithm 4 gives a short outline of this process.

Algorithm 4 Force-directed placement

```
for i = 1 to iterations do
  for v ∈ V do
    F_v(i) = Σ_{(v,u)∈E} F_a(u,v) + Σ_{v∈V} F_r(u,v) + F_g(v) + Σ_{(v,u)∈m} F_m(u,v)
  end for
  for v ∈ V do
    p_v = p_v + t.F_v
  end for
  cool(T)
end for
```

For implementation, the temperature *cool* function was set to $cool(T) = T * 0.95$ which achieved satisfactory results. Using this function, the system converged in about 50 iterations. The value of constant k_s, which represents the optimal distance between u and v in $F_a(u, v)$ and $F_r(u, v)$, is obtained by $k_s = \sqrt{A_{page}/\sum_{v \in V} A_v}$. Here, A_{page} is the area of page and A_v is the area of v. The value of gravity constant k_g is chosen from the range of $[0.01 \ldots 0.2]$. The default values of parameters in $F_m(u, v)$ are set to $k_m = 1$, $\alpha = 1$ and $\beta = 1$.

Force-directed approach produces a satisfactory layout in many cases, an example is shown in the screenshot figure 5.9. This example uses no template and the presentation order of the content is irrelevant. The equilibrium state achieves a non-grid layout where elements are semantically grouped as a result of attractive forces without overlapping.

To give designers control over the design of final layout, templates can be integrated to the force-directed algorithm to impose layout restrictions. In a template-based approach, the layout engine invokes force-directed algorithm to compute the layout for selected regions of the page. Under this scenario, the template is first adapted to the screen size using constraint-based optimization, the adapted template is then used by the force-directed

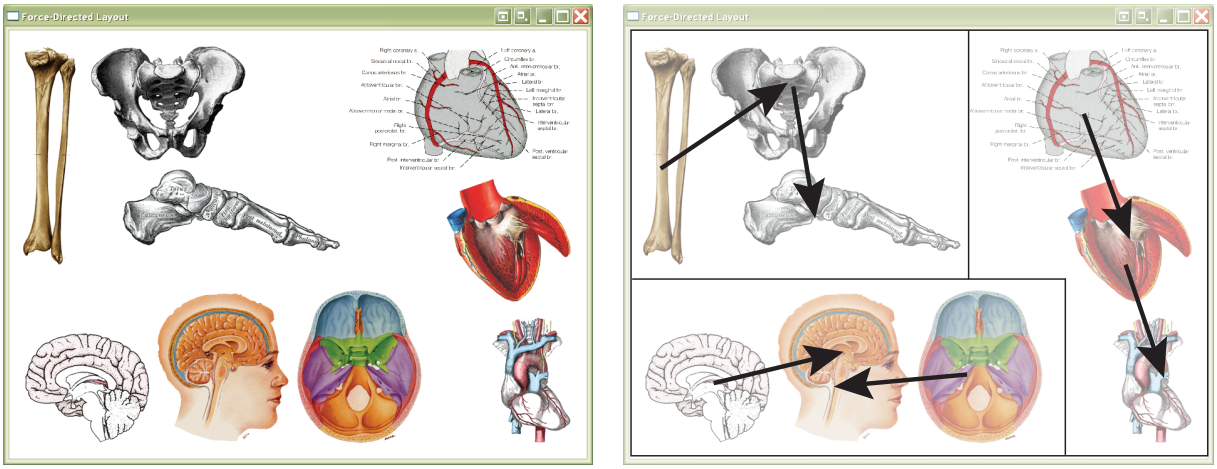


Figure 5.9.: *a)* Result of force-directed placement. *b)* Illustration of forces. Local spring attractive forces, denoted by arrowed lines, are established between related elements. Global spring repulsive force (not denoted here) is established among all pairs of elements to avoid overlapping, and also to keep the elements within the page boundary. Global gravitational attractive force (also not denoted here) is established among all elements. Spring magnetic force is not taken into consideration in this example. The layout was run using following parameters: $k_s = 30$, $k_g = 0.1$, $iterations = 50$ and $T = T * 0.95$.

approach to compute the placement of content inside template regions. Subsubsection 5.3.3 examines how constraints can be integrated to force-directed placement algorithms.

5.3.2.2. Pressure-Directed Resizing

Force-directed placement part only modifies the positions of the elements in the layout. In order to manipulate the size of elements, a pressure model is introduced in the force-directed document layout. According to Boyle's law, volume of the gas increases when the pressure decreases at constant temperature and vice versa, i.e., the volume decreases when the pressure is increased. This relationship between pressure P and volume V is given as $P \times V = constant$. Two types of pressure models are considered.

First, a pressure model based on *inner* and *outer* pressure forces taken from GUI window manager [LES95] is adapted in the document layout engine.

- The more relevant a content element, the higher the inner pressure, and vice versa;
- The more free space on the screen, the lower the outer pressure, and vice versa;

More pressure inside the element than the outer pressure leads to an enlargement of element and decrease in inner pressure and to rise of outer pressure. Conversely more outer pressure than the inner pressure leads to the reduction of element. Initial layout sets the

system of pressure forces in balance. At run-time, the pressure forces can get imbalanced. For each element node v, the pressure-directed algorithm computes $P_{inner} - P_{outer}$ and resizes the elements in order to minimize the difference between inner and outer pressure (see algorithm 5). For example, the *relevance* of an element might become higher due to user interaction, thus an increase of inner pressure would force the system to enlarge the element. This enlargement would exert more outer pressure on the remaining elements which would be shrunk in order to balance the system of forces. With this mechanism free space left by missing content elements can utilized by the other elements, new content elements can be added, removed and resized freely as the pressure-directed tries to meet the space requirements. This is a perfect scenario to integrate adaptable illustrations into complex layouts which impose and obey the global layout requirements. Since the pressure-directed resizing scheme goes hand-in-hand with force-directed placement scheme, layout conflicts such as element overlaps, are resolved dynamically.

Algorithm 5 Pressure-directed resizing

```
for i = 1 to iterations do
  for v ∈ V do
    {determine element u with maximal |u.P_inner − P_outer|}
    if u.P_inner > P_outer then
      enlarge element u
    else
      reduce element u in size
    end if
    update pressure values
  end for
end for
```

Figure 5.10 shows the result of pressure-directed resizing with force-directed layout. An element A is clicked by the user which leads to the internal pressure of the element. This causes the enlargement of A, and the rise in outer pressure which leads to the reduction of bigger element B with the lower internal pressure. Depending on their dimensions, the layout engine presents a higher resolution annotated version of A and a lower resolution unannotated version of B. These alternate versions are extracted from the multi-representation content tree as described in the previous chapter. For interactive illustrations, the annotations are placed at run-time by the annotation layout module, explained in the next chapter.

The aforementioned pressure-directed resizing approach relies either on the force-directed approach or templates to maintain the spatial constraints (such as adjacency, alignment)

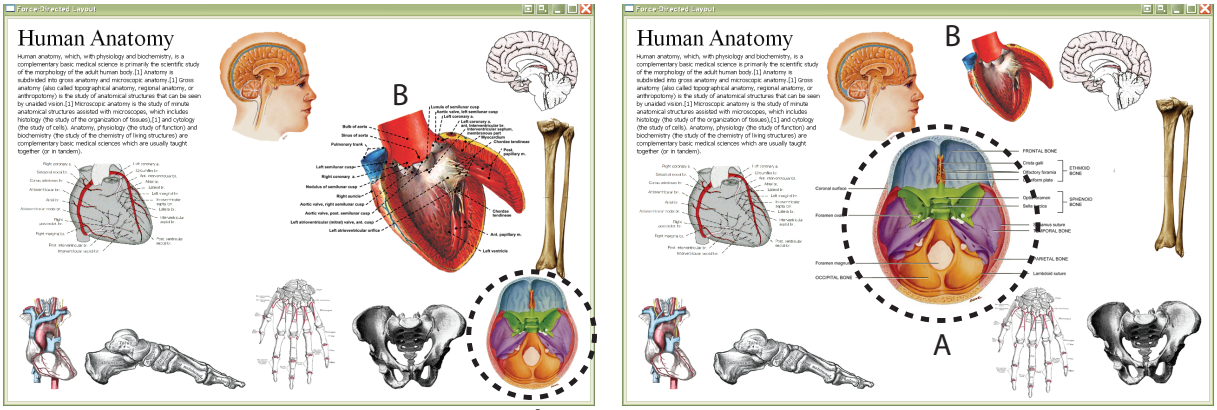


Figure 5.10.: Pressure-directed resizing. *a*- User interaction with element A increases internal pressure in A. *b*- Increase of internal pressure in A leads to the enlargement of A, which rises external pressure and reduces the size of larger element B.

among elements when the content is resized. Under this scheme, the pressure directed force algorithm is invoked separately for each individual region of the template to resize the content. In order to apply pressure-directed approach on different elements of template while obeying the adjacency constraints or topology at the same time, an alternate approach, known as *floorplanning* in the field of VLSI circuit design, is incorporated. In floorplanning, the floor is partitioned into a set of rectangles (rooms) by horizontal and vertical lines (walls). A pressure-directed model determines the shape and size of each room by maintaining topology between walls and rooms [ITK98]. According to this model, pressure force from the room r_i to each inner wall w_i is calculated which is proportional to the length of the part of w_i shared with r_i (see figure 5.11-left). A vertical wall receives pressure from the rooms adjacent to it from the right and left. Similarly a horizontal wall receives pressure from the top and bottom rooms adjacent to it. The pressure forces exerted to a wall w_i by the adjacent rooms are summed to calculate the net force F_i on w_i. A force-balancing part slides w_i to a position where F_i is minimal. This procedure is repeated for all walls until the forces are balanced. The result is shown in figure 5.11-right.

5.3.3. Integrating Constraints and Force-Directed Layout

In the previous section, we saw how different abstract and spatial constraints can be modeled using attractive and repulsive forces. Force-directed layouts do the best to display symmetries, show clustering structures and reflect semantic relationships in the layout. But a force-driven layout alone may yield inaccurate results when precise positions, alignment

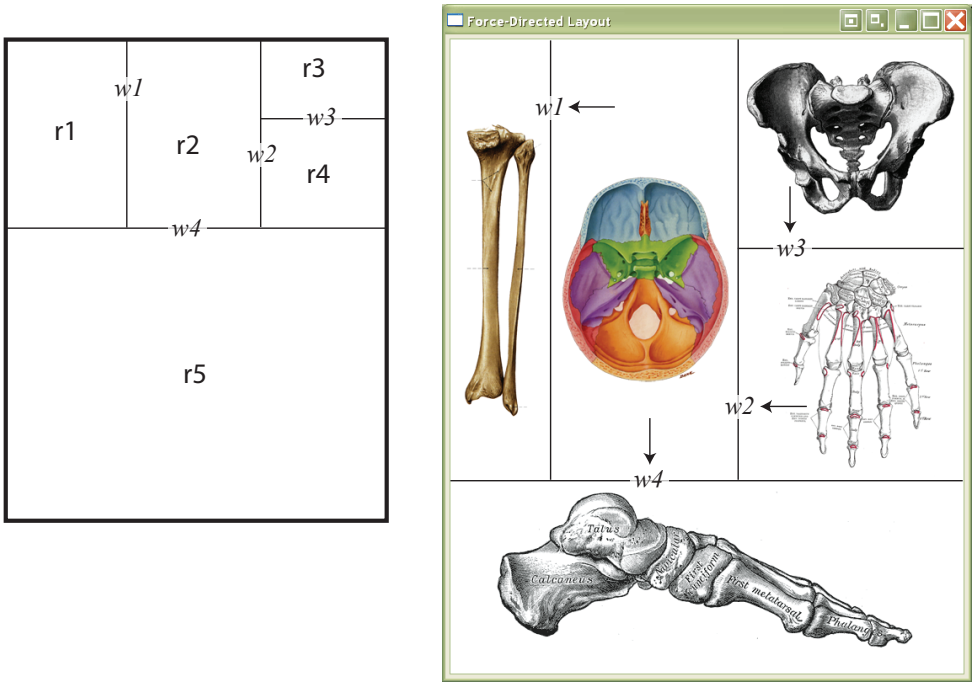


Figure 5.11.: Topological pressure-directed resizing. *Left:* The floorplan (template). *Right:* The result of topological pressure-directed resizing after moving the walls to balance the pressure forces.

or adjacency constraints must be necessary. This is due to the reason a force-directed approach seeks an equilibrium configuration—the sum of forces on each element node is minimal. For complex layouts where the final document layout must fulfill the design constraints, the force-directed layout needs to be integrated to constraint-based layout. Such an integration may be a challenging task as they both compete each other.

In this thesis, the constraint solver is employed to resolve the following types of constraints:

- *absolute constraints:* A node is fixed at its position to fulfill an absolute position constraints, and/or a node is constrained with an absolute size.
- *relative constraints:* The position of a node in relation with others in order to fulfill the adjacency, alignment and linear constraints. For relative size, the size of a node's dimensions are constrained in relation with others.
- *group constraints:* to group a set of node elements.

For an absolute position constraint, an element node is nailed at its position and is not affected by the force-directed algorithm. To fulfill relative position constraints in the force-directed layout, so called *rigid sticks* [WM96] from force-directed graph drawing, are incorporated. A rigid stick represent a position constraints in the layout. During

force-directed configuration, the two nodes (v_1, v_2) connected to the rigid-stick move together like a rigid element and thus the constraint are kept in tact. This is done by averaging the forces on v_1 and v_2 nodes and moving both nodes together in accordance with the averaged force. In the document graph model, two nodes v_1 and v_2 are attached to rigid stick, if they are restricted by *alignment/equality* or *adjacency* constraints. For linear inequality constraints, the rigid sticks are introduced only if displacing a node will violate the constraint. For group constraints, nodes are assigned into groups which are treated as a single entity. The integrated page layout approach which unifies the forces, pressures and constraint-based layout in a single algorithm is given in algorithm 6. The outer loop of the unified algorithm, **for** $i = 1$ *to iterations*, replaces the outer **for** loops of force-directed and pressure-directed algorithms.

Algorithm 6 Integrated Force-Directed Page Layout

Input: Document graph $G = (V, E)$ with set of element nodes V, and relationships E. Set of design and viewing constraints.
Output: New positions and sizes for elements V
for $i = 1$ *to iterations* **do**
1. Calculate repulsive and attractive forces on each node
2. Connect rigid-sticks to enforce linear equality/ inequality constraints
3. Distribute forces on nodes on the basis of rigid sticks
4. Move nodes to new positions
5. Pressure-directed resizing
6. Resolve constraints using constraint-solver

end for

During pressure-directed part, an absolute constraint on size prevents the system to change the size of a node, however the presence of node does give rise to the external pressure. And for a node v_1 whose size is constrained in relations with another node v_2, initial size of v_1 is set in proportion to v_2, and the internal pressure of v_1 is set equal to the internal pressure of v_2. Consequently, any pressure/size change in v_1 over the time would have a relative effect on the pressure/size of v_1.

An example which combines template-based constraints with forces and pressure given in figure 5.12. The integrated approach allows the layout engine to extend the the capability of force-directed algorithm while satisfying the design constraint in precise fashion.

Though the force-directed algorithms take frame coherency into account and try to stick the elements to their old positions in the previous layout, it may be inevitable to update the layout and move the elements to new positions and resize them or switch to a different

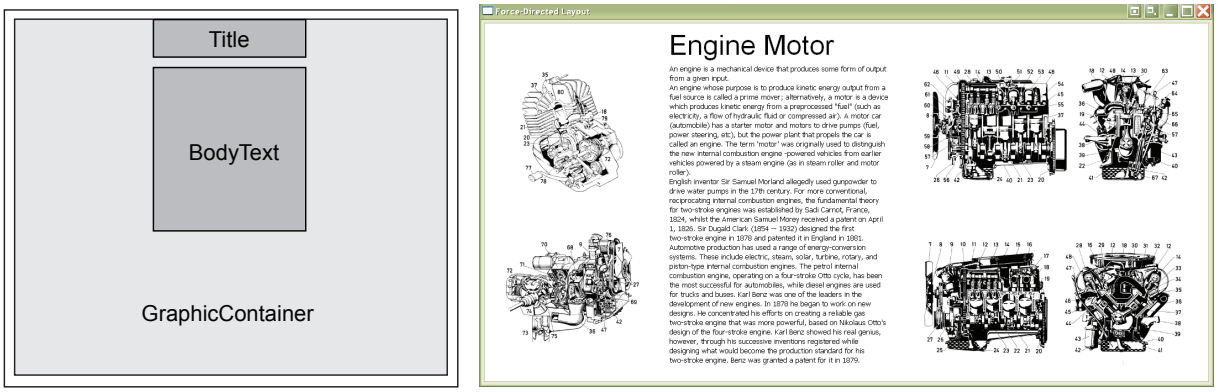


Figure 5.12.: Force-directed layout using a template. *Left:* The design template. `Title` and `BodyText` elements lay on top of `GraphicContainer` element in the template. *Right:* At run-time, the integrated force-directed algorithm respects the spatial constraints of the template. Content is first flowed inside the `Title` and `BodyText` elements in the template, and their sizes are adjusted. The remaining space of `GraphicContainer` element is used by the force-directed algorithm to place a set of illustration elements. The number of iterations were set to $iterations = 50$ for this example.

design template for the sake of optimal layout. Such visual discontinuities are eased out using *slow-in slow-out* animation technique [FvDFH90].

5.4. Pagination

Once the layout engine has adapted each valid template using constraint-based optimization, the next task is to choose an optimal sequence of page templates to layout the document. In each chosen template, the content is flowed and force-directed algorithm is applied to layout the content. Pagination of content becomes an important layout task when the document uses fixed size pages instead of a single large virtual surface that can be scrolled. As discussed in chapter 2, the goal of pagination is placing text and figures on pages in such a manner that each figure appears close to, but not before, its text reference. Computing a globally optimal pagination has been proved NP-hard, therefore in practice other measures such as number of page turns necessary for reading the document are proposed [Pla81]. These objective functions are usually optimized by dynamic programming and take multiple passes over the input. Though a multi-pass pagination [BKKW95], also called as *offline pagination*, produces superior page-breaks, the multi-pass approach can be slow for interactive documents where response time is critical.

Another concern with multi-pass pagination is that it requires that the whole content is available to the layout engine at client's workstation. In the networked environment, this

would mean that whole content is first transferred from server to the client before the layout process begins. This would further delay the document delivery process. Fairly large document would add burden to a display device with limited memory.

Even if computation and space burden is taken out of equation, there is yet another reason why pre-download of all content may not be a suitable choice. As discussed in chapter 2, one characteristic of online document that distinguishes it from traditional documents is that a digital document is served with the dynamic content. Since the goal is to provide the clients with the information on basis of context and user interaction, document content cannot be predetermined before the user begins to interact with it.

In order to compute a fast pagination without having to pre-scan all content, an *online pagination* is preferred in this work. An online pagination is a single pass algorithm, that takes only pass over the document in order to compute page breaks. It is fast enough for real-time interactive applications because it handles only one page at a time. A type of online pagination algorithm, known as *first fit algorithm* [BKKW98] is widely used by many applications, noteably TEX and LATEX. Adaptive document framework presented in this thesis adapts first fit algorithm, a type of online pagination algorithm, to assign content to multiple pages. A flow chart of this algorithm is presented in figure 5.13.

This algorithm considers a page as a series of line boxes. Text is poured into these boxes one by one according to the font type and size as indicated by the stylesheet. As soon a reference to graphic is encountered, the figure is added to the figure queue. At the end of each paragraph (or a line for that matter), the first figure in the figure queue is placed on the current page, provided there is enough space left on the page to accommodate the figure. When a page is completely full, a new page is inserted. Otherwise, placing line boxes and figures on the page is resumed. For a new page, the layout engine again resorts to the template selection process mentioned in section 5.2 and chooses an appropriate template to paginate the content. For a multi-column page, the pagination procedure is execute from one column to another, and a new page is inserted once all columns on the page have been filled with the content. Figures can be placed both in the slots defined in the template as well as they can float in the text regions as inline figures. Inline figures are scaled proportionally to take the whole column width.

Note that this flow chart only shows a greedy *first-fit* approach for text placement. The advantage of greedy line-breaking algorithm is that it takes $O(n)$ time to layout n words. Also, the same greedy approach is able to compute the text placement for both rectangular

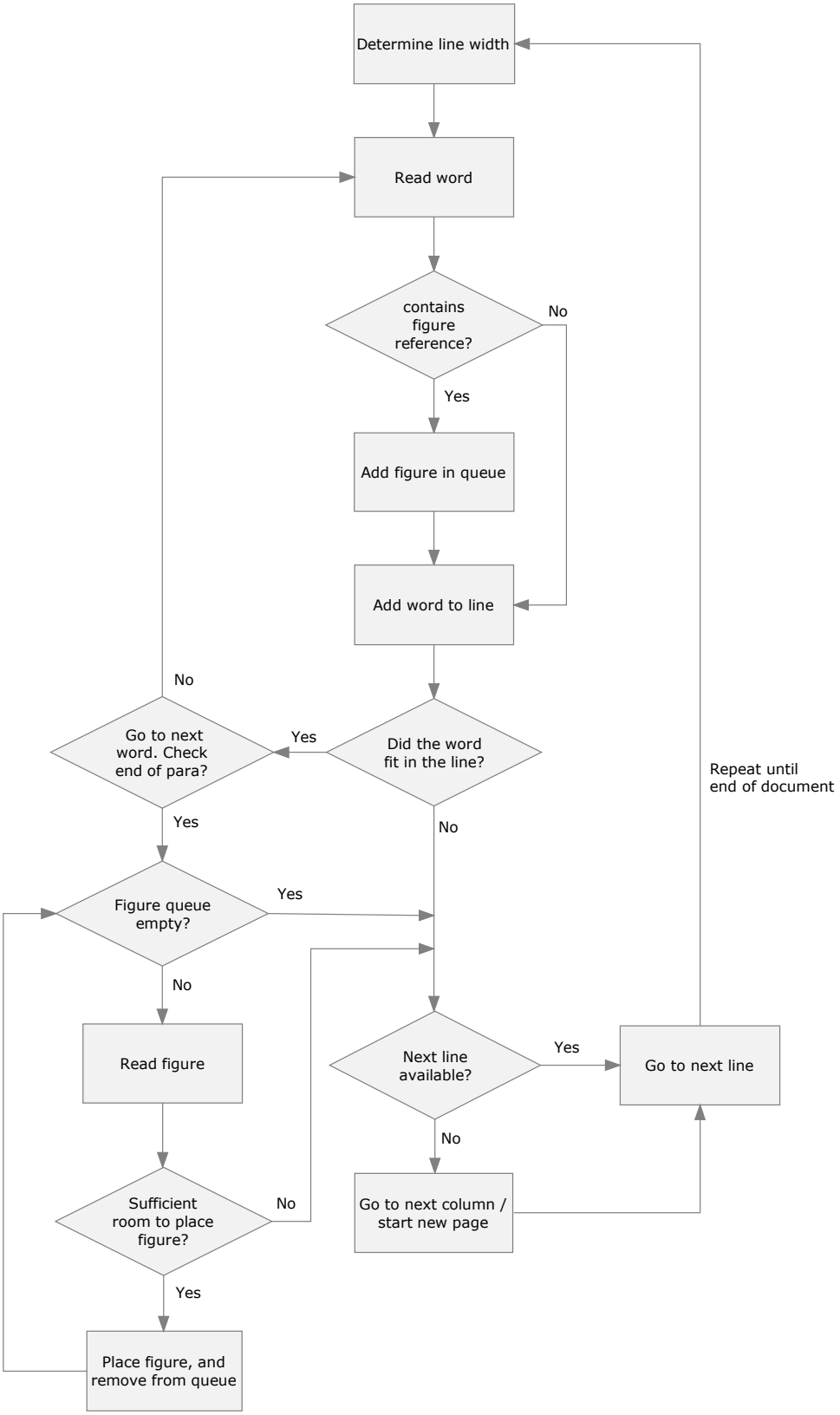
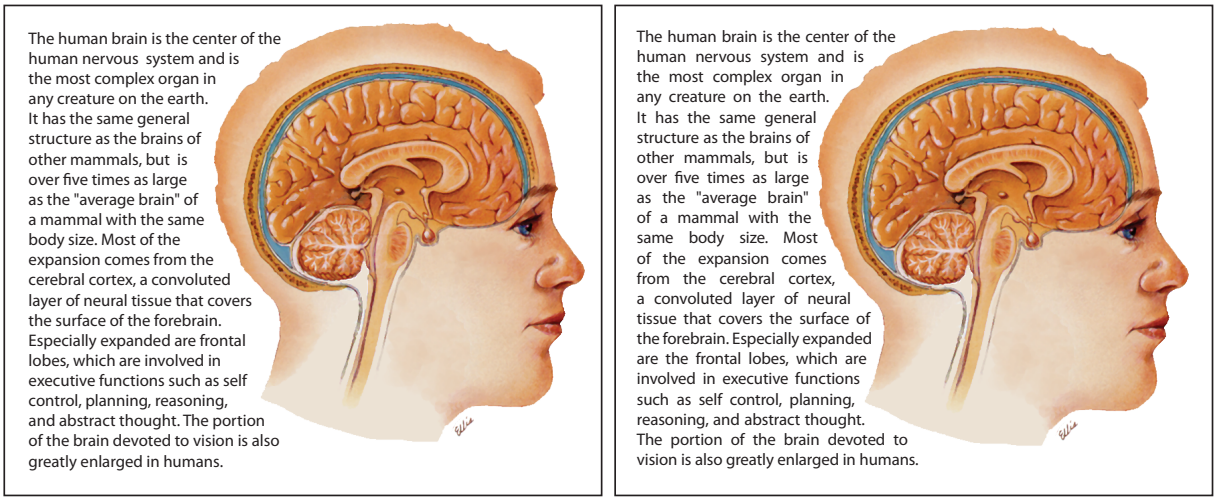


Figure 5.13.: Flow chart of single-pass pagination.

regions as well as non-rectangular regions, because it does so one line at a time. The only requirement for non-rectangular regions is that minimum line width for any height must be larger than the width of the widest word.

A *total fit* approach for optimal line-breaking, such as used by LaTeX [KP81] (see subsubsection 2.3.2.1), can easily replace the greedy line-breaking algorithm in the pagination algorithm. A total fit algorithm takes a paragraph as input and computes an optimal line division of a paragraph at once. The resulting text lines can be then laid on the page one by one in the fashion mentioned earlier. Dynamic programming implementation of optimal line breaker costs $O(n^2)$. If the text block is non-rectangular, in which case all lines in the paragraph may not have the same width such as when text wraps around a figure, optimal text placement is achieved in two passes. The first pass determines the width of each line in the text block. The second pass then computes the final placement of words in the variable-length text block. Figure 5.14 shows the result of greedy line placement versus *justified* total-fit line breaking. Though, total-fit line placement looks more aesthetic, professionally designed tutoring material like Visual Dictionary [Doc02] prefer a rugged style for text wrapping around annotated illustrations, the results of which are similar to greedy placement.



(a) Greedy line breaking

(b) Total-fit line breaking

Figure 5.14.: Text placement in non-rectangular text blocks.

Clearly, when time and space efficiency are not considered as the key issues, and all content is known in advance, dynamic multi-pass pagination [BKKW95] can be used to achieve optimal pagination.

5.5. Experimental Considerations

Figure 5.15 presents several screenshots of a document which is adapted to various screen sizes. In this figure, each example uses the same template yet the difference in screen size leads to different results. The layout is achieved by combining the constraint solver with the force-directed algorithms. The layout parameters for this example were set to: $k_s = 20$, $k_g = 0.1$, $iterations = 50$ and $T = T * 0.95$. The k_s plays an important role in the force-directed placement, as it sets the distance between nodes. The default value of k_s was based on formula at page 84 which takes into account the total free space on screen. Consistent results were obtained from this value. However, sometimes, the default value of k_s does not yield visually pleasing results particularly when the majority of content elements in the layout have elongated shapes and different aspect ratios. In this case, it required manual tweaking of parameters to get better results. The values of k_g and k_m were chosen experimentally. A value of $k_g > 0.2$ tends to produce relatively compact layouts, while a lower value such as $k_g < 0.01$ yields relatively sparse layout. Experiments suggested that the results were fairly consistent when the value of *iterations* was selected from a range of 50–100, and the annealing cooling rate was set to $T = T * 0.95$. Increasing the number of iterations and decreasing cooling rate such as $T = T * 0.99$ or vice versa didn't indicate significant improvement in the results. With approximately 50 iterations, the performance of integrated layout approach was consistently interactive for up-to 30 repositionable elements in the layout.

5.6. Discussion

This chapter presented a new approach for automated document layout adaptation. The constraint-based layout is an appropriate choice to describe spatial constraints which must be obeyed in the document layout. But a constraint-based template adaptation is only the first step toward adaptive document layout. It may not be possible for the designer to specify a precise design if it includes unknown compound content and when the layout is dependent upon the content-driven semantic constraints. A novel force-directed approach is presented that translates the semantic constraints to manipulate the size and positions of the elements. The results show that force-directed approach performs very well to layout the rich illustrative material in the document. A unique feature of force-directed layout is that it generates symmetric non-grid design merely from content semantic structures. A

variety of constraints including in particular but not limited to, overlapping, correlation, frame coherency, resizing items, etc. can be easy modeled through forces and pressure. Although spatial constraints can be modeled using forces, accomplishment of precise results for a complex layout would require fiddling with attractive and repulsive force parameters. It also requires the us to add logic to the way forces would be summed together. Therefore, a unified constraint and force-based approach is presented that combines the constraints provided at design-time with the content-driven forces calculated at run-time to produce adaptable grid-based and non-grid layouts for illustrated content. The unified approach also reduces the burden of template authoring as it will allow the designers to specify essential constraints in the template and let the layout forces do the rest of the work during run-time.

After describing an approach for automated document layout adaptations in this chapter, the next part of the layout engine named as the annotation layout module, which computes the placement of annotation for individual illustrations, is elaborated in details in the following chapter.

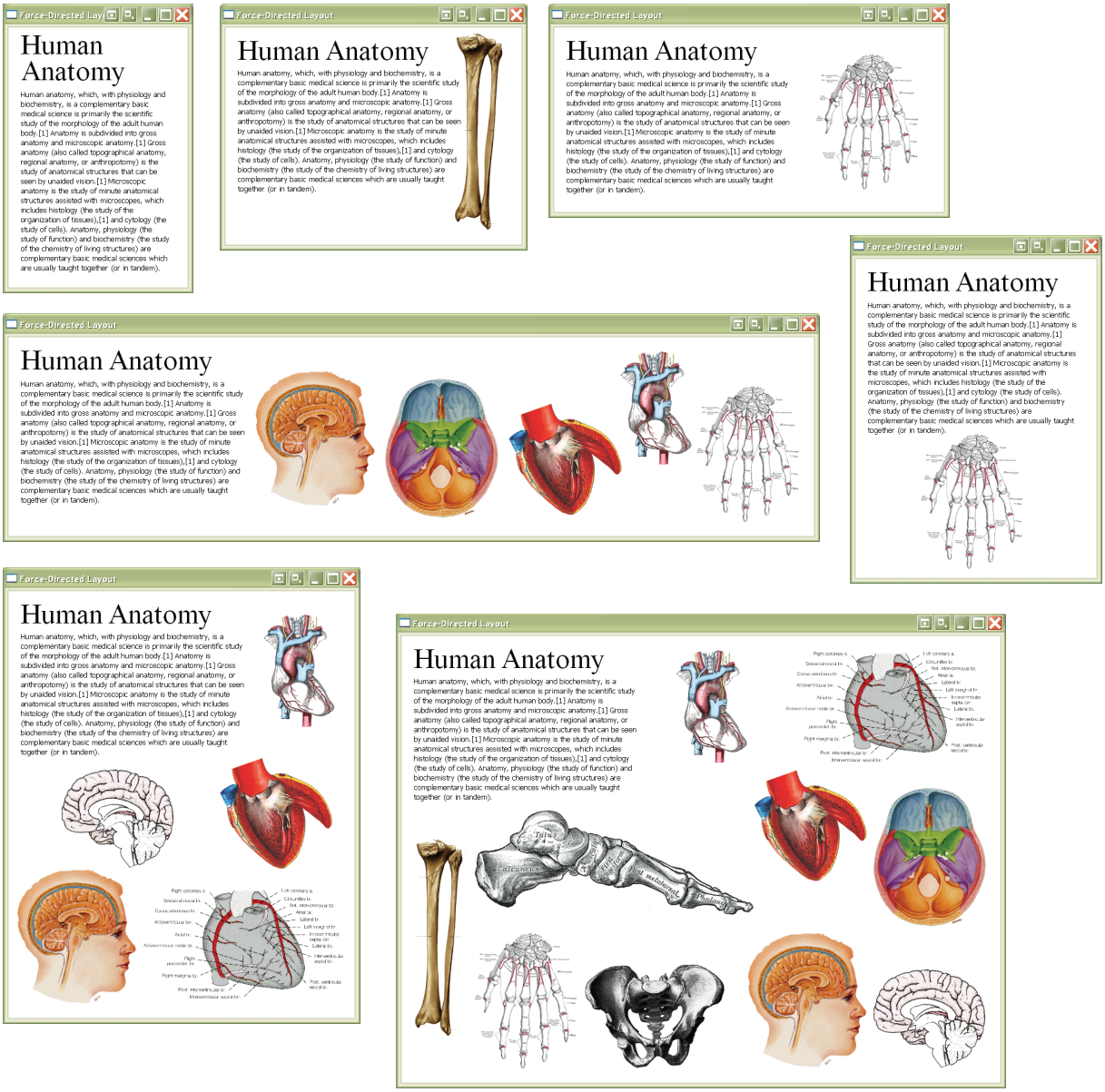


Figure 5.15.: Examples of force-directed layout at various screen sizes using the same template. Note that the semantically related elements are place together in three groups—heart, brain, and bones.

Chapter 6

The Layout Engine II: Annotation Layout

In order to illustrate instructive materials, documents are frequently enhanced with additional textual or visual elements. Human illustrators introduced a number of techniques to achieve this task—labels, insets, legends, and figure captions. This way of tagging visual objects with textual information is known as annotating or labeling. An efficient annotation serves several *functions* in parallel: (i) it introduces references of unknown terms by linking text to their related visual elements, (ii) it gives verbal descriptions for unknown visual objects, and (iii) it focuses the attention of the viewer on important aspects of the illustration. Thus labeling is potentially exploited within learning materials, where many unknown terms in a domain-specific or foreign language have to be conveyed in parallel.

Nowadays, modern interactive tutoring documents incorporate renditions of 3D models which enable the viewer to explore the complex spatial configuration of technical devices or organic structures. Learning efficiency of these on-line tutoring systems can be greatly improved by integrating annotations in the dynamic visualizations.

This leads us to a question: "where to place annotations in the picture?" Determining the best placement of labels solely from the criteria through optimization procedures could be infeasible due to two reasons: (i) There are infinite number of candidate positions where the labels can be positioned on the picture; thus, evaluating best positions for all labels is computationally hard. (ii) Several aspects those are important to designing hand-made illustrations cannot be precisely modeled through optimization techniques which produce inferior layouts. Computational hardness of layout problem aside, the automated layouts

have to also follow different annotation rules (conventions) employed by human illustrators. These rules help the human illustrators to achieve better readability and visual balance in illustrations. Various layout styles can be seen in hand-made illustrations which can be classified according to their common properties. The observation leads to an idea of using these layout styles for an automated label placement in order to improve the usability and appearance of layouts. Moreover, solving the layout problem according to the requirements imposed by individual layout styles dramatically eases the layout task.

The extraction of labeling styles from hand-made illustrations and their implementation is acutely described in this chapter. Based on the analysis of a corpus of illustration books, section 6.1 classifies a set of annotation layout styles. Functional and aesthetic attributes of layouts are discussed in section 6.2. Section 6.3 presents a powerful, unified approach to produce automated annotation layouts for interactive 2D/3D illustrations. The approach is inspired from hand-made illustrations and takes into account the functional and aesthetic attributes while placing annotations. A collection of annotation layout algorithms are proposed in section 6.4 to annotate objects in dynamic illustrations. Some extensions and applications of the automated annotation approach are given in section 6.5. Section 6.6 concludes the chapter.

6.1. Analysis of Annotated Illustrations

This section describes the manual analysis of annotation layouts in hand-drawn illustrations. A number of anatomical text books, atlases and visual dictionaries were examined to classify various layout styles for different annotation types.

6.1.1. Types of Annotations

Human illustrators employ a number of techniques to integrate textual and visual information which can be classified into two broad categories:

- **Image overlay** methods enhance the images with annotations. Textual annotations can be overlaid directly on the co-referential visual objects, we call them *internal annotations*. Internal annotations are used for fairly large sized objects. Alternatively,

annotations can be placed on the *negative space*[1] of the image. In this case, they are called *external annotations*. External annotations are connected to their associated visual objects via *connecting lines*. The endpoint of the connecting line which sits on top of graphical object is called *anchor point*. Both internal and external annotations, empowered by image overlay method, serve as quick visual reference to name the objects in the image (see figure 6.1). Other examples, though more descriptive than visual, are text insets inside objects as internal annotations.

- **Non-overlay** methods present the reference text without altering the image. They are used to provide in-depth details of the topic. The most common example are figure captions, placed below the image. Figure captions are usually short, spanning over a couple of sentences, and give a brief summary of the image. The supplementary paragraphs (wrapping or non-wrapping), along side the image, refer to the figure and illustrate the subject matter in further details.

- **Hybrid** methods are a mixed of overlay and non-overlay annotations. An example is legend key—a supplementary table accompanying an illustration which gives the meaning of overlaid symbols in the image.

When compared to non overlay method, which is an indirect way of referencing, image overlay techniques instead take a direct route by merging verbal and graphic elements together in the same space. That's the reason why textbooks commonly use internal and external annotations, a form of image overlay technique, to name the pictorial elements in the illustrations. This aspect is clarified by Mayer through *spatial contiguity principle* [May01]. According to this principle, when corresponding text and pictorial elements are presented near each other on the page or screen, the learners do not have to waste cognitive resources as a consequence of switch between text and image. Thus the readers can hold the both representations in their working memory and establish association between them, which will in turn help them in mental integration of visual and textual elements.

Therefore, in practice, image overlay techniques dominate the area of annotation layout and are complimented by non-overlay techniques. Important issues in this regard are how the text annotations will be placed inside the picture so that the relationship between text

1 Negative space, in graphic design [Whi02], is all space that is not primary subject. Usually, it's the background area in the image that is not occupied by type and graphic elements.

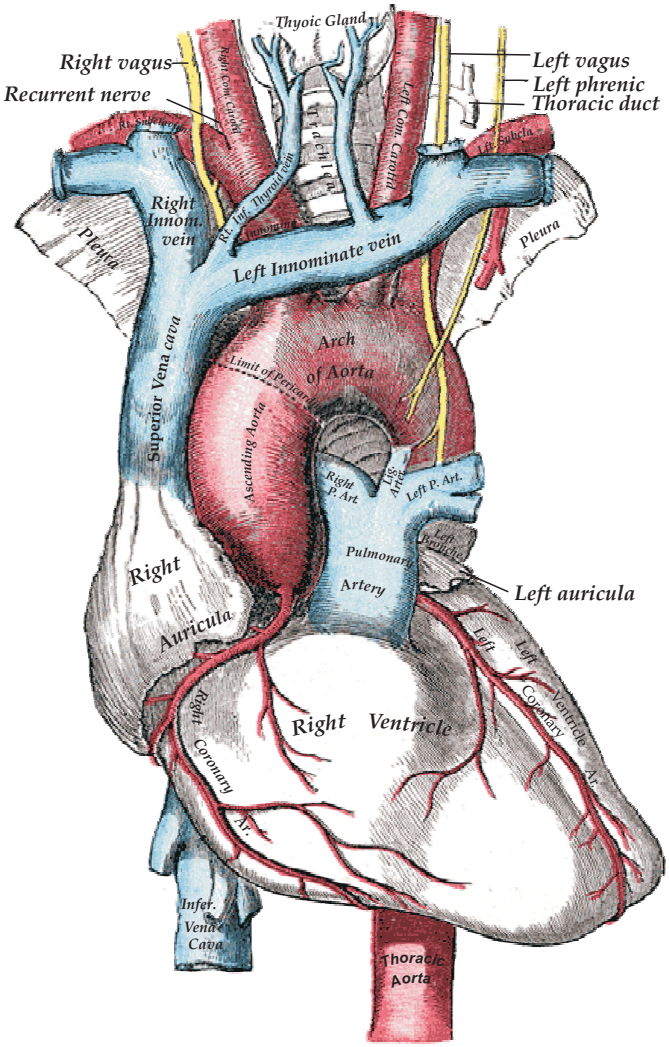


Figure 6.1.: Illustration with internal and external labels. (Source: [Gra18])

and image elements becomes clear, and making sure that the text does not hide the key features of the image.

Internal annotations are intuitive because they directly overlay the objects which makes their association unambiguous. However, there are some limitations. First, an internal annotation placement may not be possible when the amount of text does not fit within the graphical object. Second, if the shape of graphical object of concern is thin and irregular with sharp curves, the internal placement of text may have poor effect on its readability. Third, internal annotation may obscure features that are otherwise important in the image. Due to these reasons, illustration in practice prefer external annotations over internal annotations. External annotations only occupy the background space and do not overlay primary objects. This ensures a better legibility of text and also preserves

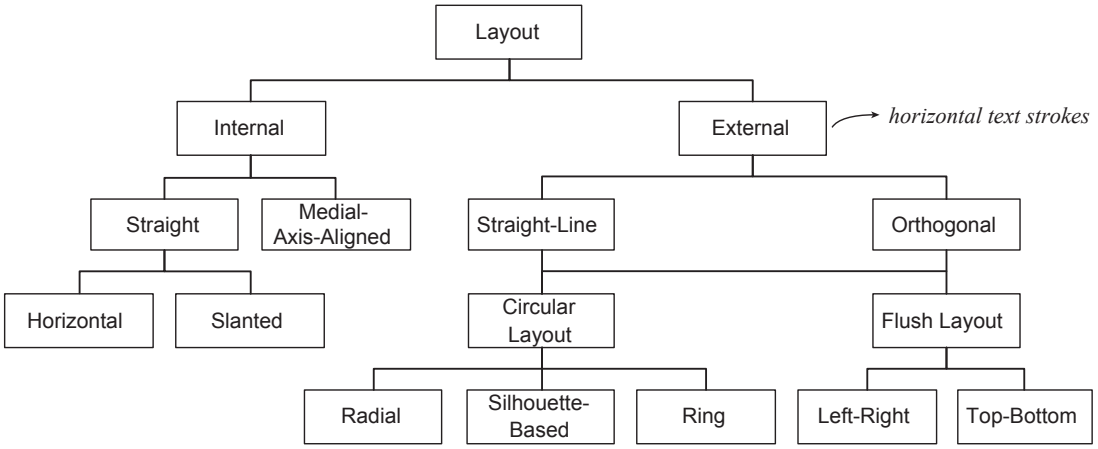


Figure 6.2.: Annotation layout classification.

the visual details of the image. This also allows to extend the external annotation to any size on negative space, provided there is enough room in the image. Sometimes, it is even better to use both internal and external annotations to label different parts of an illustration. When used together, internal annotations overlay their reference objects, that are relatively large sized, and external annotations occupy the negative space, and used for fairly small sized objects.

6.1.2. Annotation Layout Styles

In practice, hand-made illustrations employ a variety of different labeling styles for internal and external labels. The choice of label layout styles reflects both space restrictions imposed by the overall layout as well as subjective preferences of illustrators. Figure 6.2 presents, based on our informal study of corpus of illustrations, an overview of hierarchy of layout styles for both internal and external annotations [AHS05, HGAS05].

Internal annotation styles

In illustrations, internal annotations are used less frequently as compared to external annotations. The shape of internal labels depends on the *annotation path* along which the text string is laid out. The annotation path can be either a straight horizontal line, or straight but slanted, or curved along the medial axis of the object to suggest the shape and extent of objects (see figure 6.1). As the internal labels overlay the object areas, text color and background colors of text must have sufficient color *contrast* to be readable. To ensure the legibility, labels can be rendered with halo around the typeface, making the

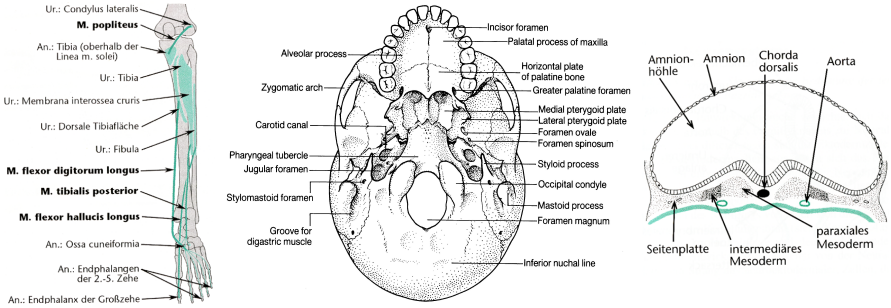


Figure 6.3.: Flush layout styles for external annotations: *Left image:* flush left layout based on area alignment. *Center image:* orthogonal-lined flush left-right layout based on shape alignment. *Right image:* flush top-bottom layout. (Source: [MM00], [Rog92] and [MM00]).

letters clearer. Alternatively the color of letters can be chosen to provide the maximum contrast to the overlaid part of surface.

External annotation styles

External annotations are how the majority of all illustrations are labeled. Reference lines establish the visual link between labels and anchor points that are marked at the assigned objects. There are different *line styles* with which reference lines can be drawn. They can be bended at orthogonal angles or appear as straight lines. For *alignment*, annotations can be arranged in different styles, vertically or horizontally aligned (flush layouts), or arranged in a circular manner around the graphical objects.

In flush arrangement, annotations are forced to lie in specific regions of negative space. These regions are simply designated as left, right, top or bottom, their location with respect to the graphical subject. In flush left-right style, labels are placed only in the left and/or in the right regions. Similarly in flush top-bottom style, labels are placed only in the top and/or in the bottom regions. In flush layouts, labels are aligned to other labels in the same region to bring symmetry and ease flow of reading. A few examples of flush layout are given in figure 6.3.

In circular arrangement, annotations are placed around the graphical model in a circular fashion. This arrangement is particularly suitable when the graphical model has a quasi-ellipsoid shape. In radial layout style, the labels are positioned around the graphical model, but label placement and the connecting lines are radial with respect to a common origin, usually the center of graphical model. A somewhat similar style is ring layout that sets the labels at uniform angles around the graphical model. The most common form of circular arrangement is silhouette-based. A silhouette-based layout style places the labels near the silhouette boundary of the graphical model and minimizes their

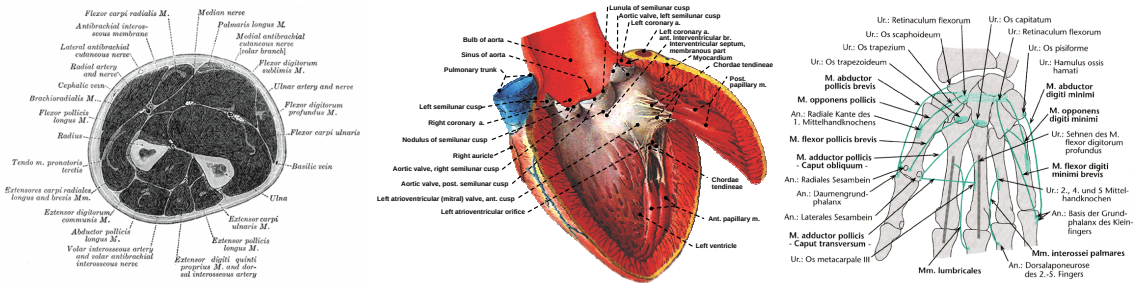


Figure 6.4.: Circular layout styles for external annotations. The annotation are placed around the shape of the subject in circular fashion. (Source: [Gra18], [SPP97] and [MM00]).

distance to the corresponding anchor points thus keeping the length of connecting lines short and potentially increasing the reading efficiency of illustration. Figure 6.4 provides some examples of circular layout styles in hand-made illustrations. Usually, the quality of illustration goes hand-in-hand with the number of annotations. Large number of annotations make it difficult to produce a good layout. It also impedes reading.

6.2. Functional and Aesthetic Labeling Attributes

The main challenge for human illustrators while placing labels is to consider a number of conflicting requirements such as readability, unambiguity, media capabilities, publishing costs, and subjective preferences. Therefore, manual label placement becomes a time-consuming task that takes considerably longer to create complex high-quality annotation layouts. The following subsections discuss various functional and aesthetics attributes for annotation placement in both traditional and interactive media [AHS05, HGAS05, GAHS05b]. Such attributes can be used by the automated process in order to achieve the optimal annotation layout.

6.2.1. Legibility

Legibility of the illustration is a degree at which annotations are readable based on appearances. Typographic decisions, such selection of face type and size, etc. determine how legible is the text in the annotation layout.

- *Internal annotations:* Area and linear features have to provide enough space for internal labels, otherwise they are considered as point features and labeled externally.

The text strokes could be either horizontally aligned or they should be drawn as smooth as possible, as a high curvature reduces text readability. Steep text strokes should be avoided as well. If the path length exceeds the required length to display text strokes, then an appropriate path segment is searched. Moreover, at least a minimal contrast between letterings and its local environment has to be guaranteed.

- *External annotations:* External labels should neither overlap one another, nor the connecting lines and visual objects. Moreover the connecting lines should not cross each other. At least a minimal contrast between meta-graphical objects (connecting lines and anchor points) and its local environment has to be guaranteed.

6.2.2. Unambiguity

The layout should guarantee that the viewer can easily extract the co-referential relation between labels and their associated visual objects.

- *Internal annotations:* Text strokes should be placed over salient regions of line or area features and must not leave the object's interior. Moreover, text strokes should not be placed at very narrow areas.

- *External annotations:* Labels should be placed as close as possible to their co-referential visual objects. Anchor points should placed at salient location on their corresponding visual objects. Moreover, the anchor points of different objects should not be located too close to each other. The length and the number of bends in the connecting line should be minimized. Connecting lines should not cross each other and must not occlude another points.

6.2.3. Visual Occlusion

Annotations and the visualization share the same presentation area. Therefore, it is important that the annotations are integrated in a way, that their placement does not cause visual occlusion of the important parts of illustration and does not interfere with other annotations in the layout.

- *Internal annotations:* Annotations should not occlude key parts of the visualization. When presented together with external annotations, internal annotations should not cover the connecting lines.

- *External annotations:* External annotations should not occlude the primary graphical objects. Connecting lines should not cover internal labels and key locations of primary objects.

6.2.4. Aesthetics

Visual aesthetics in an annotation layout should not stand solely for beauty. The main purpose of the annotation layout (*form*) is to communicate in an effective manner (*functionality*). This argument refers to the famous dictum of the architect Sullivan *"form ever follows function"* which became one of the most influencing guidelines in industrial design due to Mies van der Rohe and other artists from the Bauhaus school. Thus, the quality of a annotation layout should be judged according to functional attributes, such as legibility, unambiguity and visual occlusion. In other words, aesthetics attributes in the annotation layout are mainly dominated by functional aspects of label placement. However, the research in automated document layout (e.g., [HNJ+04]) gives several aesthetics guidelines that could be adapted for the annotation layout as well. The *regularity* principle induces that annotations elements should be aligned and spaced in a regular fashion. Also the annotations should use the same font type as well as size and the dimensions of the labels should be regular. The *uniformity* principle suggests that the annotations should be distributed uniformly in the presentation space.

A major factor in aesthetics is the *balance* [TB64] which means that the elements on the page should be placed to achieve a sense of the equilibrium or wholeness. The balance of layout depends on the *visual weight* of each element (in our case the annotation), and the *visual center* of a page. Visual weight is determined by size and grey-value of annotation, and visual center is a point slightly to the right of and above the actual center of a page. In *centered-balance*, the center of visual weight is at the visual center of a page, and in *left-right balance* the visual weight of the left side is matched by the visual weight of the right side. Similarly in *top-bottom balance*, the visual weight of the top side is matched by the visual weight of labels on the same horizontal position at the bottom side.

- *Internal annotations:* There are not specific aesthetic attributes for internal annotations as such, other than that the internal annotations should be placed in a way without clutter the illustration. Arbitrary high curvature of annotation path should be avoided. The principle of regularity and uniformity apply. Dense visual clustering of internal annotations should be avoided.

- *External annotations:* The annotation should be aligned with respect to one another (horizontal or vertical *alignment*) or along the silhouette of visual objects. Regularity, uniformity and balance principles are very importance in external annotations. The labels and anchor points should not be densely clustered in the illustrations.

6.2.5. Interactivity & Frame Coherency

The dynamic characteristics of digital documents introduce additional requirements for the interactive illustrations. Annotation layout must be updated if the underlying visualization changes. To guarantee a smooth interaction with the presentation, the annotation layout should be computed at interactive rates, at least 15 frames per second. The label layout should be adaptive, i.e., it should reflect contextual requirements. This could be achieved by considering the interaction context and user-specific requirements within the label selection and by displaying dynamic contents within the labels.

In interactive environments, computing the annotation layout independently in each frame, without considering the previous layout might generate visual flickering. This involves sudden appearance and disappearance of annotations and the effect of motion when layout elements shift to other positions. If the changes in layout occur instantaneously, users can lose their focus from the primary objects. Therefore, such layout changes should be either avoided or smoothed out in the presentation.

- *Internal annotations:* Typically, an internal annotation follows the shape of graphical object and is placed along the skeleton of object. Even minor changes in the shape of object may result in drastic changes of the skeleton. Therefore, those segments of the skeleton which minimize the displacement of text strokes are extracted. However, this feature should be disabled to achieve more aesthetic text strokes for static images. More salient locations, i.e., larger area features or longer line features, offer a greater stability in the layout.

- *External annotations:* The displacement between subsequent anchor points and label positions should be minimized. Moreover, the number of permutations in the label sequence and the assignment of labels to the different label area should be minimized.

6.3. Dynamic Annotation Layout Framework

In the previous two sections, the focus of the discussion were layout styles and the specification of various functional and aesthetic attributes of annotation layouts in the domain of hand-made illustrations in print media documents and interactive media. This section uses these specifications as the basis for the development of a framework to automatically annotate 2D images and 3D visualizations in the context of interactive multimedia documents. As discussed in section 2.2, adapting a document may require adjustment in content, this also applies to the graphic elements in adaptive documents which need to be dynamic. Due to varying contextual and viewing requirements, illustrations created with a particular goal in mind can rarely be used in contexts different from the original. From the presentation point of view, the static illustrations annotated with text labels cannot be shrunk down and scaled up arbitrarily, because that can affect the legibility of text. Therefore, it is often essential to adapt the given illustrations according to new communicative functions and to other external requirements imposed by the layout of parent document. This would require that the text and graphics in the illustrations are handled separate from each other. Adaptive illustrations would offer new possibilities for online documents by adjusting the layout and content displayed within textual annotations. In the broad scenario, the document layout system would assign tasks and impose contextual constraints, such as designating the maximum area of annotated illustration and objects of interest, and the annotation module would carry out annotation layout task by fulfilling the designated constraints.

The computation of annotation layout can be considered as an optimization problem, as the placement of an individual annotation might have a global effect on the quality of layout. But, even a simpler problem—finding an optimal layout for the point-feature labeling—is computationally intractable NP-hard [FW91, MS91], requiring exponentially long running times, and therefore not feasible. Achieving annotation layout styles, particularly for complex 3D visualization, that meet the specifications derived in section 6.2 make the

problem become even more complex. The degree of difficulty of annotation problem is highlighted by the lack of effective approaches presently available to solve the problem.

As discussed in section 2.3.3, these approaches, [PRS97] as an example, either implement a small subset of those layout styles found in scientific or technical documents, or are error-prone estimation of annotation placement based on rough shape approximations [BFH01], or they rely on user interaction to achieve an appealing annotation layout [LAS04, RSHS03]. Hence, an approach is needed to address the problem effectively which can offer us a wide variety of layout styles at interactive rates.

For adaptive document representation, it is important that the layout mechanism based on annotation styles is adaptive—it provides transitions between internal and external annotations. Ideally, the method should be developed in a way that it can be used to annotate both 2D and 3D visualization seamlessly. The solution needs be pluggable to the generic document layout framework, as we have discussed in chapter 3. Furthermore, other existing applications, such as 3D browsers and illustration programs, should be able to integrate the solution and benefit from it, without constraining the visualization and interaction as such.

Therefore, the objective is to compute annotation (layout) configuration based on a set of styles and various functional and aesthetic attributes discussed in sections 6.1 and 6.2. An annotation configuration specifies all parameters of a chosen layout candidate. For a set of visual objects and their associated annotation, it computes: (a) those annotations which should be contained in the illustration, (b) their classification (internal vs. external), and (c) all layout-specific annotation parameters.

These aspects are reflected in the proposed adaptive annotation module [GAHS05a] as shown in figure 6.5. The approach relies on several heuristics to determine layout styles for both 2D and 3D visualizations. The annotation framework consists of a set of three *buffers* that store at each pixel, shading information (*frame-buffer*), segmentation information (*ID-buffer*) and distance to objects information (*distance-buffer*). With the use of these buffers, there is no further distinction necessary between 3D and 2D, and same techniques can be applied to annotate both type of visualizations. By analyzing the ID- and distance-buffers, the annotation module decides for each object whether to use internal or external annotation for it and then computes the placement of internal and external annotations in the image. The annotation layout process is further guided by the following contextual parameters:

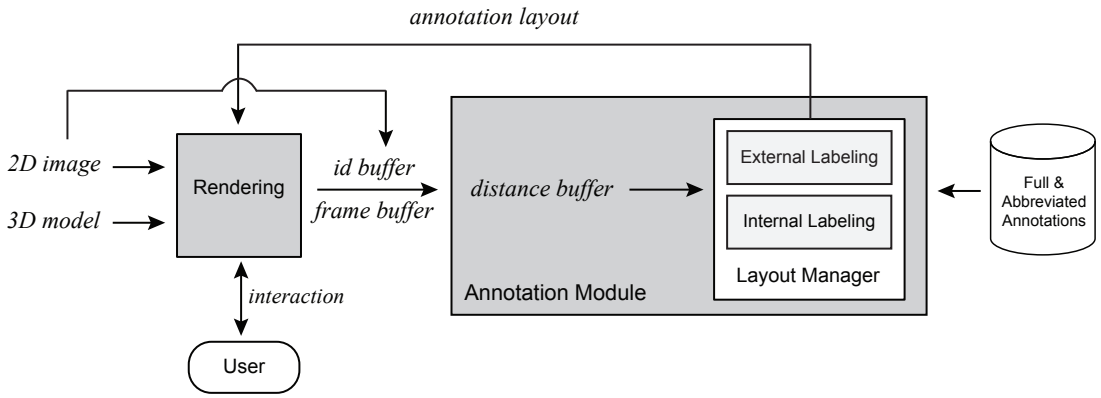


Figure 6.5.: Adaptive annotation layout framework.

- *Relevance of visual object:* is determined through user interaction and task model (cf. section 4.4). The relevance of an object determines whether the corresponding annotation should be included in the layout or not, and the level of detail for annotation. Here, level of detail refers to the amount of content presented in the annotation.
- *Amount of free space:* the amount of space available inside objects and outside affects the decision whether to set as internal or external annotation, which layout style to choose as well the level of detail for annotations.
- *User preferences:* the annotation style actually preferred by the user.

The annotation module obtains content from an *annotation table* that stores abbreviated and detailed forms of annotation text. The final annotation layout is sent to the application module and can be rendered on top of the frame-buffer. In the following section the creation of buffers and other parts of annotation system are explained.

6.3.1. Creation of the Buffers

For the purpose of being independent of the type of visualization, 2D or 3D, a set of three buffers are used. These buffers store graphical model's shaded projection, segmented view of the visualization, and distance to object information in the visualization.

Frame-buffer

First kind of buffer used by the annotation system is a frame-buffer which is the image containing a complete frame of data displayed on screen. For 3D, the system renders 2D

shaded projection of 3D scene which is captured as frame-buffer (see 6.6-a). In case of 2D, the frame-buffer is identical to the 2D illustration on screen.

ID-buffer

Second of kind is ID-buffer that represents segmented view of all visible graphical objects. ID-buffer, also known as *color-coded* image, assigns each pixel of the frame-buffer a unique color-ID value (see 6.6-b). This buffer is required by the annotation module to analyze the visibility of objects and to determine the space locations for annotations.

In order to compute ID-buffer for 3D visualization, the rendering module internally renders the model using flat shading. All geometric objects in the ID-buffer are rendered with unique colors. The background is assigned a unique color too, for example pitch black. In that way, ID-buffer, at first, separates foreground objects from each other and also from the background.For each frame-buffer image that the user sees on the screen, behind the scenes an ID-buffer is generated by the rendering module. The process of rendering ID-buffer consists of the following steps:

- Assign new material properties (*emissive color*) to geometric objects,
- Switch off lights,
- Switch on flat shading,
- Render and grasp frame buffer and store it as ID-buffer.

For 2D visualizations, the creation of ID-buffer is a bit different though, as the geometry information may not be explicitly indicated in the image. One possibility is to take the frame-buffer (the 2D visualization) and apply *image segmentation* methods [JKS95]—based on various geometric properties such as color, intensity, or texture—to locate the objects in the image. The automatic segmentation problem belongs to the field of *computer vision*, and is a research area on its own. It is a difficult problem which may require the users to tune various threshold parameters on individual basis and the results can be fuzzy at times. However, for annotating illustrations, the ID-buffer must be accurate and the associations of objects with the annotations must be clear. Therefore, a manual segmentation of objects is unavoidable. Within the annotation framework, we assume that each input 2D image that needs to be annotated comes bundled with the corresponding ID-buffer.

An analysis of ID-buffer reveals vital information to the annotation layout module, such as which objects are visible, extent and shape of objects and negative space. This information

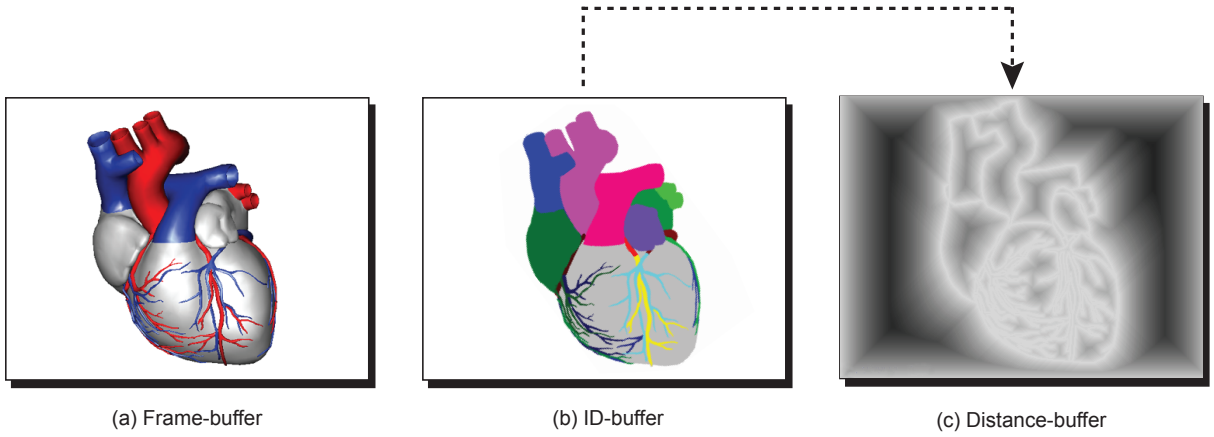


Figure 6.6.: Frame buffer, ID-buffer, and distance-buffer. In distance-buffer (c), dark pixels correspond to high distance values.

is used by annotation placement algorithms to compute the annotation paths for internal labels, anchor points and locations for placing external labels. Moreover, the unique color-ID is used for each object to pick-up the annotations from the annotation table as explained in the next section.

Distance-buffer

A crucial part of the annotation layout process involves determining the extent and location of free space inside and outside objects to place internal and external annotations respectively. Internal placement requires following the shape of the object and keep a safe distance from the object boundary. External annotations need to be arranged at negative space, i.e., at pixel locations with background color in ID-buffer. In order to calculate the extent of free space, we need to compute the distance from background pixels to the foreground pixels. Furthermore, for external annotations we need to evaluate the distance of annotations to their corresponding object. In the end, it turns out that most of the decision in the annotation layout process revolve around the determination of distances [DD06]. This is where the third kind of buffer, *distance-buffer*, comes into play.

Distance-buffer, also known as *distance transform*, *distance map* or *distance field*, is an image where every pixel has a value corresponding to the minimum distance from the nearest *obstacle pixel*. In our case, the obstacle pixel is a boundary pixel in ID-buffer, which either marks the boundary of a feature object, or the edge of the image. Distance transform is a useful tool in computer vision [JKS95], and has many applications in model matching, shape analysis of objects in an image [Dan78], and in computing medial axis skeleton of an object in the image [GCW94, SM87]. Since the approach in this thesis uses

digital images, 2D images or 2D projections of 3D model, to annotate visualizations, a distance transform becomes a natural choice. As we will discuss later, distance transform and the skeletons based on distance transform will be used to compute the annotation path for internal labels as well as position of anchor points and external annotations. Because the distance transforms in the framework are computed from ID-buffer, they provide fairly good accuracy as far as the visibility and shape of the objects projected on screen are concerned. Another advantage of distance transform is that there exist fast efficient methods to calculate *distance fields*, which take only two linear passes over the image [RP68], and a similar method called *chamfer distance transform* described in [Bor86]).

During working with digital images, there are various metrics to chose from as an approximation for measuring the distance between two pixels, for example *Euclidean*, *Chess Board*, and *City Block (Manhattan)* distance metrics (see figure 6.7). Chess board and Manhattan metrics are faster than Euclidean metric (because the latter involves square root operation), but less accurate. To get better results, yet in an efficient way, another metric, known as *Pseudo-Euclidean* metric, exists which is an integer approximation of Euclidean metric [Ad88], that can used in conjunction with 2-pass distance transform algorithm [RP68] to efficiently compute the distance-buffer. In the first pass, the algorithm runs from top-down and left-right. A distance value is calculated for each pixel o by comparing distance value of its neighborhood. A minimum distance value m from the neighborhood n_i is located. If m is smaller than the current distance value at o, then m is incremented by the distance between o and n_i. The second pass runs from bottom-top and right-left in the similar fashion, albeit with a different neighborhood operator (see figure 6.8). In each pass, the minimum distance to the boundary of segment is updated for each pixel through propagation. The total computation time for distance transform is $O(n)$ with n number of pixels in input image. A screenshot of distance transform is shown in (see 6.6-c).

6.3.2. Annotation Table

The co-referential relation between textual annotations and graphical objects is established using a relational table. The illustrator or 3D model designer defines the textual annotations at various levels of details, ranging from abbreviated notations to detailed descriptions, that are stored in a knowledge base external to annotation module and 2D image/3D

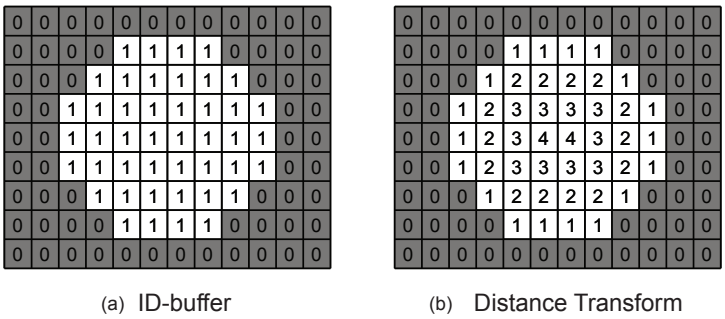


Figure 6.7.: City Block (Manhattan) distance transform when applied to a binary ID-buffer. To help identify the negative space, the pixels with zero value have been marked with as dark.

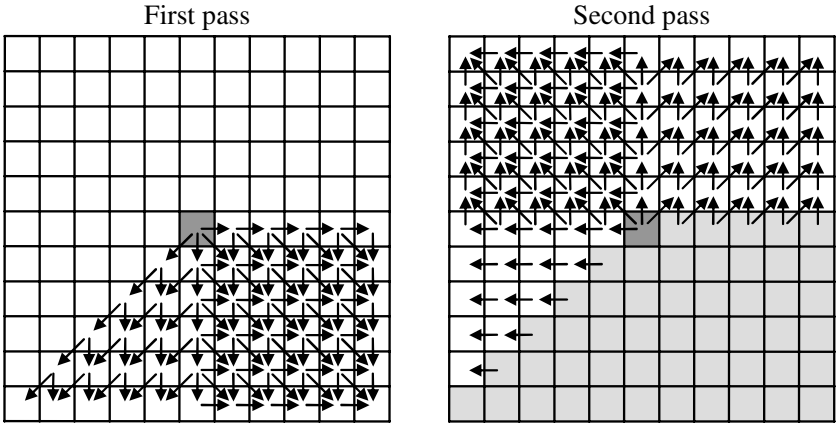


Figure 6.8.: Two-pass distance transform algorithm using *chess Board* distance metric. In this example, the distance values are propagated from a single background pixel in the middle. The arrow lines show where the minimum distance value comes from. When there are multiple arrows pointing to a pixel, all neighbours have the same value. Source: [Bai04].

model content database. The annotation module accesses the annotations for each object by mapping its color-ID value in the ID-buffer to the ID key in the annotation table (see Figure 6.9). In this decoupled approach, the annotation table forms a bridge between the annotation layout module and underlying 2D images/3D models. One set of annotations can be reused by different versions of visualizations and vice versa. The information in annotation table is represented in an XML file that is loaded and stored dynamically.

A task model (cf. section 4.4) can define which annotation table to use for the current context or which fields in the annotation table to use in order to annotate objects.

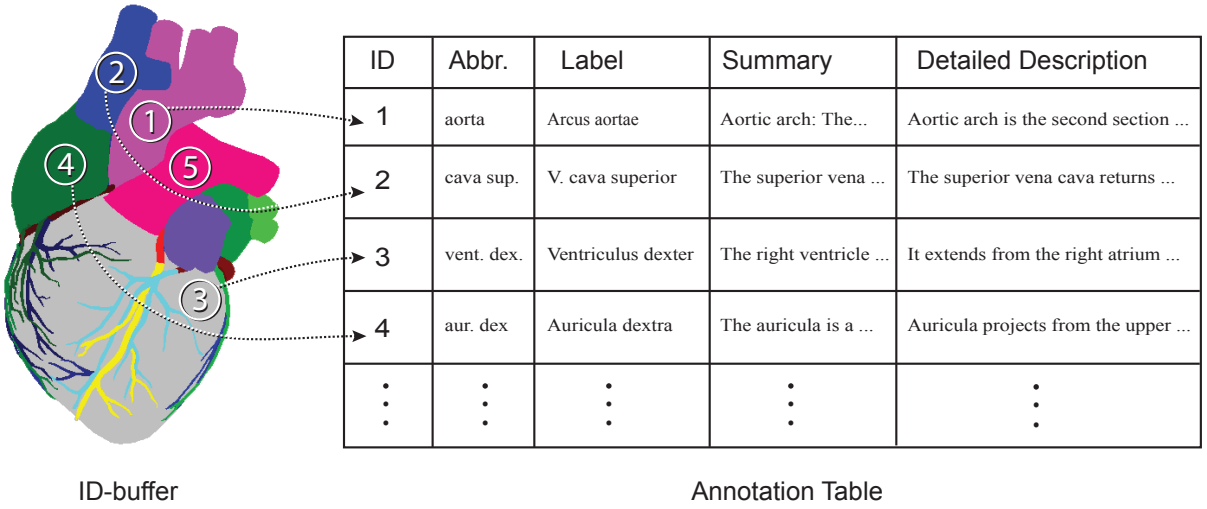


Figure 6.9.: The color-ID of the objects in the ID-buffer is used to uniquely identify the textual annotations in the annotation table.

6.4. Annotation Placement Methods

This section presents strategies to determine the placement of internal and external annotations in the visualizations. The automated annotation layout approach uses the set of buffers, as discussed in subsection 6.3.1, and realizes the layout styles that were derived in corpus analysis section 6.1. Whereas the annotation algorithms in this thesis primarily focus on external label placement methods published in [AHS05, Ali05, FLH+06, HAS04], internal annotation algorithms were developed in cooperation with Goetzelmann and integrated together in the annotation layout framework as published in [GAHS05a, HGAS05, GAHS05b, GGA+06, GGA+07].

6.4.1. External Annotations

For objects that are not internally annotated, annotations are placed outside the objects, on the negative space. In order to establish the visual reference between annotations and target objects, connecting lines are used. Determining the location of endpoint of connecting line which overlays the object, anchor point as it is called, is very important task because it establishes connection of object with the annotation. Since the annotations are secondary elements in the illustration, generally it is better to first decide where to place anchors and then proceed with the placement of text annotation than the other way around. To do that, the object segmentation information can be obtained from the

ID-buffer, that tells us which objects are visible. ID-buffer and distance-buffer are then used together to determine the locations of anchor points and placement of annotations, as we shall explain in the following.

6.4.1.1. Anchor Computation

For external annotation, the annotation module has to compute which locations serve as anchor point for the objects of interest. Since an anchor point is intended to support the identification of visual object and its distinction from the remaining objects, its position is crucial to prevent co-referential mismatch. Typically, illustrators use one anchor point for each object they wish to label. Infrequently, one can also find examples where a label is connected to a complex shaped object via multiple anchor points. But, this makes annotation layout task even more challenging. On the technical side, it becomes difficult to avoid line intersections when there are several labels in the layout each connected to multiple lines. When used more often, multiple anchor points / object may also put extra burden on readers to identify multiple text- graphical associations. On the basis of these factors, we conclude that within the scope of this thesis, it is suffice to deal with one anchor point per object in the annotation layout.

From the corpus analysis of hand-drawn illustrations, we infer some guidelines for determining good anchor points:

(a) *Validity:* Anchor points must overlay their corresponding objects.

(b) *Salience:* Place anchor points at the biggest part (segment) of the objects. Furthermore, place anchors at the most internal locations of objects.

(c) *Separation:* Avoid clusters of anchor points.

The fields of computer graphics, computational geometry and image processing provide numerous approaches to extract the geometric features of 3D and 2D shapes. Anchor points can be computed either from 3D geometric model, or from segmented image (ID-buffer). The methods for computing anchor points in 3D [PR98, SCS04] are of static nature. They compute anchor points from geometric mesh, and thus there is no guarantee that the anchor point will be visible when projected on the screen. Using ID-buffer that always contain visible parts of the objects, the 2D anchor computation methods could be chosen to find valid anchor points.

Simple 2D operators, such as *center of mass* and *bounding rectangle center*, can be used to determine anchor points, but if the objects have 'L' or 'U' shape, neither the center of the bounding box nor the centroid guarantees the valid anchor point.

To get valid and the most (salient) internal location in an image segment, 2D *voronoi skeleton* [PS85] of a segment can be computed from segment boundary pixels in ID-buffer. The skeleton is the locus of the centers of maximal inscribe circles. Once the voronoi skeleton is computed, the *largest empty circle* inside the segment boundary) on voronoi skeleton is located. The center of largest empty circle is a location that maximizes its distance from the boundary pixels; hence a best candidate (in terms of salience) for placing anchor. The largest empty circle can be computed in time $O(n\ log\ n)$ for n number of voronoi sites. However when working in image space, with a large number of segment boundary pixels, computing voronoi skeleton can still be a slow process. Sub-sampling of boundary pixels can be speed-up the process [Sch89] but may also yield wrong results.

To compute the anchor points in accurate yet efficient manner, instead we can use distance-buffer that gives us pixel distance information from the object boundary. For each unique colored object in the ID-buffer, the distance buffer is masked by that object and the location of highest distance value is selected as anchor point for the object (see Figure 6.10). The pixel location with the highest distance is the one which is furthest away from the boundary pixels inside object, thus fulfilling validity and salience criteria.

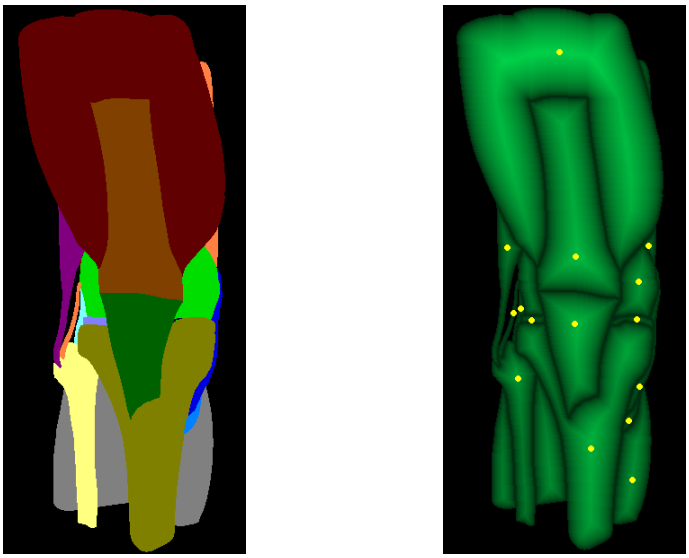


Figure 6.10.: Color-coded image (left) and the distance-buffer (right) overlaid with anchor points (shown in yellow). The dark pixels have small distance values, and bright pixels have large distance values. Anchors are placed at the brightest pixels of the objects on distance-buffer. Note that only one anchor point is chosen for each colored object (even if has multiple segments).

In order to fulfill the separation criteria, that states that the anchor points should not be placed close to each other as it may cause ambiguity problems in the layout, a *subtractive function* is applied to the distance-buffer. After computing an anchor point, the subtractive function centers itself around that anchor point's location c and decreases the values of pixels p in distance-buffer D in its influence region R.

$$D(p) = D(p) - f(|p - c|) * k \quad ; \text{for } p \in R$$

where f is a non-negative decreasing function and $|p-c|$ is the distance between pixel p and c, and k is a scaling factor. The result of this function is that the distance values around the freshly computed anchor point are reduced and the next anchor points are selected at locations with lesser distance values and further apart from the previous computed anchor points.

Figure 6.11 represents this idea in 1D. Green and red curves denote the distance values for two objects. After placing an anchor point at peak c for the green object, the subtractive function (blue dotted line) is centered at c , and applied over distance image which forms a valley in R. The nearby peak 1 is cut down, and the new candidate anchor points for the red object are peak 2 and 3. Finally, peak 3 is selected as anchor position since it is now the highest. Hence, the anchor for red object moves away from it's previous location at peak 1 which was nearer to the green object's anchor point. The subtractive function will be next performed at peak 3. A greedy algorithm finds out the peaks for anchor points one by one, and modifies the distance image each time. Figure 6.12 presents the result.

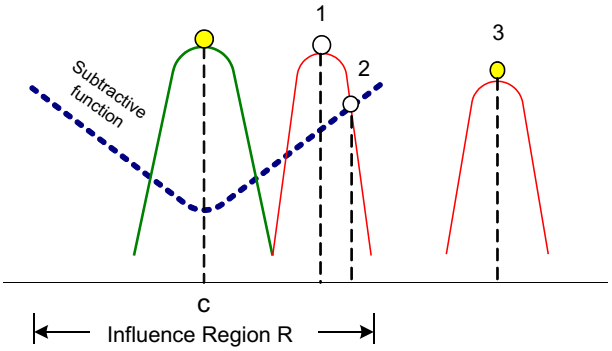


Figure 6.11.: Separation of anchor points

Searching for the highest distance value for each object takes $O(n)$, where n is the number of pixels in the distance-buffer. Object segmentation of ID-buffer and distance-buffer is computed only once for all objects throughout the layout process, and takes $O(n)$ time, which is also very affordable in our framework.

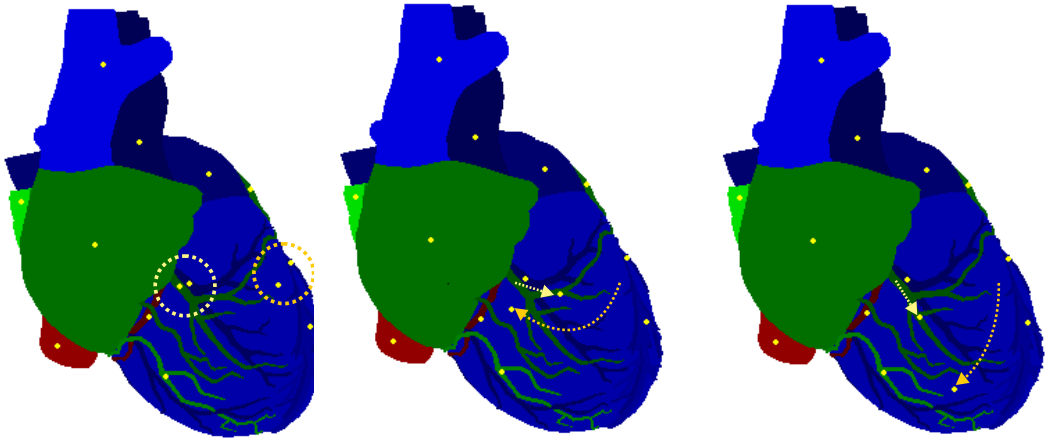


Figure 6.12.: Example of separation of anchors. Left: Anchor points which will be separated are encircled. Center: Separation of anchor points when the influence region is smaller. Right: Separation of anchor points when the influence region is larger.

6.4.1.2. Generalized Layout Algorithm

The annotation layout approach in this thesis, inspired from stylistic conventions by human illustrators, implements various layout styles for automated annotation placement in interactive illustrations. A set of annotation layout heuristics were designed that are guided by various functional and aesthetic attributes from hand-made layouts. As will be shown later in result, this intuitive approach not only improves the quality of layout, it is also fast and adaptable to dynamic nature of online documents.

The algorithms, proposed in this regard, work on greedy approach for *initial annotation placement* and subsequently improve the results in order to meet the layout criteria. The conflicts (e.g., line intersections, label overlaps and label clustering) which arise during the initial label placement are resolved at later stages. This modular approach narrows down the problem scope at each step and makes it possible to carry out the computation in real-time. Algorithm 7 describes the steps which are generally required to determine the annotation positions in the picture.

As a side note, since the external annotations are placed outside the reference objects, it is very important for layout styles to know the *silhouette boundary* of the graphical model to keep the annotations outside the object boundary. Due to the complex shape of objects, the silhouette boundary of graphical model could become concave that makes it difficult to find empty space around the model. Therefore, the silhouette boundary of graphical model is approximated with a simple (convex) shape. Each annotation layout style defines

how the silhouette boundary would be approximated, and how the annotations would be arranged in free space available outside the simpler boundary. In the later stages, the annotation layout can be further refined by bringing the annotations closer to the actual boundary.

Algorithm 7 Generalized Layout Algorithm

1: `ComputeAnchors:` Determine the positions of anchor points
2: `FindSpace:` Determine the extents of empty space regions
3: `Allocation:` Allocate spatial regions for label
4: `InitLayout:` Compute an initial label layout
5: `Stacking:` Stack labels to eliminate overlap
6: `RemIntersect:` Resolve line intersections
7: `Compact:` Perform layout compaction

`1. ComputeAnchors:` Anchor points are computed, as described in subsubsection 6.4.1.1, for all objects of interest in the ID-buffer.

`2. FindSpace:` The extent and locations of empty space regions around the graphical model in the picture are computed. Four *biggest empty rectangles* (*Left, Right, Top, Bottom*) around the bounding box of the model are located, which serve as regions to contain labels. The bounding box representation is used to locate the regions for initial placement of labels. During the Step. 7 *(layout compaction)*, labels would be allowed to dive into the bounding box of model to get themselves close to their corresponding objects.

`3. Allocation:` Anchor points are sorted and then divided into different *sets* depending upon the layout style. This reduces the search area as each set is treated separately during the layout process. Based upon the division of anchor points into different sets, and the location and capacity of the empty regions, the labels are assigned to empty regions.

`4. InitLayout:` The labels are assigned positions in regions according to the layout style. Labels may overlap, and connecting lines may intersect at this stage. Some layout style algorithms result in a very few or no label overlaps. Details of style specific algorithms will be given in subsubsection 6.4.1.3.

`5. Stacking:` To avoid label overlaps, labels are examined, and in case of overlap they are stacked on top of one another while keeping the average length of connecting lines to be small.

`6. RemIntersect:` To eliminate line intersections, the layout is checked to detect which of the connecting lines intersect. The labels are then repositioned to leave behind no line intersections in the layout.

`7. Compact:` If required, layout compaction can be performed to arrange the labels close to the boundary of graphical model (cf. 6.4.1.4). Layout compaction reduces the average length of connecting lines that leads to the reduction of the area of annotation layout.

Since the layout criteria can conflict with each other, following precedence is established while carrying out the layout process:

- prevent label-label overlap,
- prevent line intersections,
- choose layout style,
- maximize number of labels,
- reduce length of connecting lines, and
- perform layout compaction.

6.4.1.3. Style Specific Algorithms

Previous subsection presented a general methodology of annotating graphical objects. Different layout styles have their own requirements and they can skip or need a special behavior at particular steps of the generalized algorithm. This subsection describes which changes are needed in the generalized algorithm to implement the different layout styles proposed in section 6.1.

Flush Layout

For *flush* layouts, step 3–5 in generalized algorithm 7 are implemented as follows.

Step 3 (`Allocation`)—allocate spatial regions for labels:

(a) sort anchor points according to the x direction,

(b) choose a *pivot* point (e.g., mean or median of anchor points), and

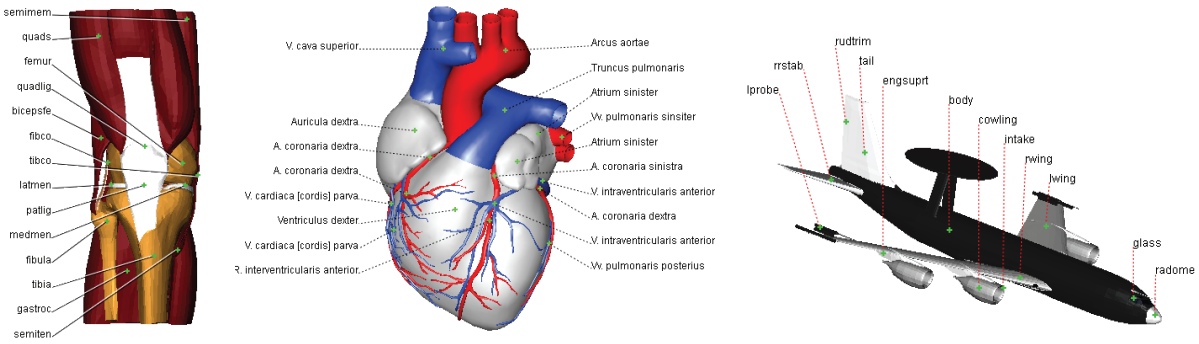


Figure 6.13.: Examples of automated annotation layouts done in flush layout style.

(c) assign labels to the left and right region by comparing their anchors with the pivot point.

Step 4 (`InitLayout`)—compute an initial label layout:

(a) assign the y position of anchor points to y position of associated labels, and

(b) justify the labels on the bounding box of the graphical model.

Step 5 (`Stacking`)—stack labels to eliminate overlap:

(a) a recursive algorithm assigns new positions to the labels to eliminate label overlaps and minimize the average vertical length of connecting lines.

For *flush left* and *flush right* layout, we assign to the pivot element a minimal or maximal value. For *flush top* and *flush bottom* layout, the previous algorithm works by exchanging horizontal and vertical directions. Later on, in top and bottom regions, labels are stacked in vertical direction. Some results of flush layout styles are shown in figure 6.13.

Circular Layout

For circular layouts, three variant, namely, radial, ring and silhouette based layouts are implemented.

Radial Layout

This layout style places annotation around the ring at positions which are radial projections of corresponding anchor points. Radial layout style modifies step 4 and 5 of generalized algorithm 7.

Step 4 (`InitLayout`)—compute an initial label layout:

(a) select a center position o (e.g., mean or median of anchor points) and an appropriate radius r for a circle C which encloses the graphical model,

(b) compute the radial projection of the anchor points on C, and

(c) align the corresponding label on this position. Labels on left-half of C are right-justified and labels on right-half of C are left-justified.

Step 5 (`Stacking`)—stack labels to eliminate overlap (as in Flush Layout).

Figure 6.14 illustrates the above algorithms.

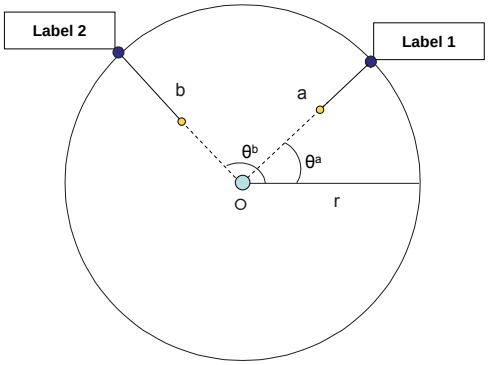


Figure 6.14.: Radial projection.

Radial projection point P of anchor point A can be obtained from:

$$P_x = r\cos\theta \ , \quad P_y = r\sin\theta$$

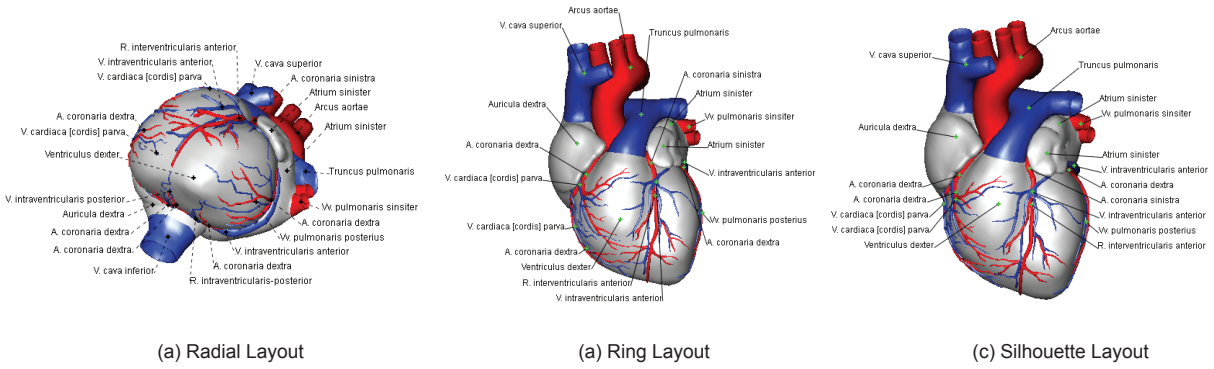
where r is the radius of circle, and θ is angle in radians formed by A with the horizontal axis centered at o.

The radial projection of anchor points produces no intersections of connecting lines. However, labels can overlap or lie very close to each other which is resolved by label stacking (see figure 6.15-a).

Ring Layout

Using ring layout, labels are placed at regular intervals on a ring which encircles the graphical model. This layout style modifies step 4 and 5 of generalized algorithm 7.

`4. InitLayout:`—compute an initial label layout:



(a) Radial Layout (a) Ring Layout (c) Silhouette Layout

Figure 6.15.: Examples of automated annotation layouts done in circular layout style.

(a) select a center position o (e.g., mean or median of anchor points) and an appropriate radius r for a circle C which encloses the graphical model,

(b) choose n evenly spaced positions P on the circle, and

(c) determine a *bijective mapping* from the label set to P which minimizes the distance between labels and their anchor points.

`5. Stacking:`—stack labels to eliminate overlap:

(a) for labels in the upper half of C, stack labels on top of each other; and for lower half of C, stack labels to the bottom of each other.

Figure 6.16 illustrates the algorithm.

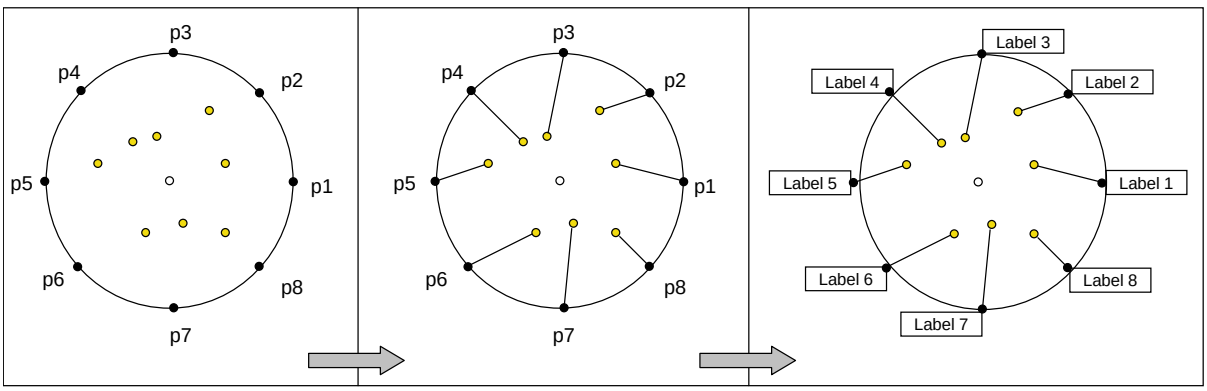


Figure 6.16.: Placing labels on ring. *Left:* Creating uniformly spaced positions p_i (filled black circles) on the circle boundary. *Center:* Computing a bijective mapping from anchor points to p_i. *Right:* Placing labels at p_i.

Figure 6.15-b shows the results of ring layout algorithm. For relatively small values of n and moderate r, there is no need to check for label overlaps, as they are already evenly spaced at the first step. When the labels have much more width than the height, as the number of labels increases, overlaps can occur at the top and bottom regions of C, and hence stacking is necessary in those cases.

Silhouette-Based Layout

This layouts sets the labels around the convex hull boundary of graphical model. Labels get the positions that are closest to their corresponding anchor points. Silhouette-based layout style modifies step 4 and 5 of generalized algorithm 7.

`4. InitLayout:`–compute an initial label layout:

(a) compute the convex hull of the geometric model and enlarge it (pre-processing step), and

(b) project the anchor points on the edges of the convex hull silhouette boundary S. For each label, choose the projection position P as label position which is closest to the anchor point (see figure 6.17).

`5. Stacking:`—stack labels to eliminate overlap:

(a) for upper half of S, stack labels on top of each other; and for lower half of S, stack labels to the bottom of each other

Orthogonal Projection of u onto v: An anchor a is projected onto every edge v of the S. The projection point P which gives a minimum length line between a and P is chosen as the label position. Other orthogonal projection lines are dotted in the figure.

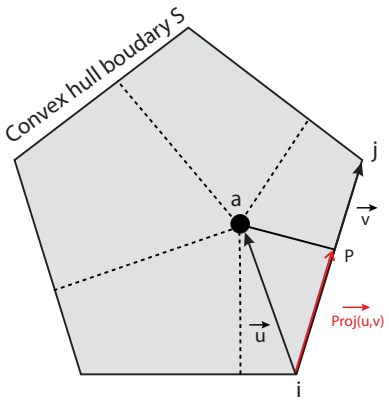


Figure 6.17.: Orthogonal projection.

Figure 6.15-c shows the screenshot of silhouette-based layout.

Silhouette based layout or radial layouts algorithms can result in an uneven distribution of annotations around the graphical model, which is fine in most cases, but can be annoying

if that leads to a huge stacking of labels on top of each other to avoid overlapping. To avoid that effect in circular layouts, and instead balance the distribution of labels to some degree, force-directed placement techniques are considered once again. Previous chapter 5 modeled a document layout problem as a graph layout problem which is solved using a system of forces. Now an external annotation layout problem is modeled as a graph layout problem. From the functional and aesthetic attributes in the section 6.2, an external annotation problem is formulated as—*place labels close to their anchor locations in the layout, but away from other labels as to avoid any overlapping*. We now discuss how attractive and repulsive forces are used to solve this problem.

Let, an external annotation layout be represented as a graph $G(V, E)$. Vertices V denote labels and anchors. Edges E denote relationships between labels and anchors, or in other words, connecting lines. The original force-directed algorithm (cf. section 28) is modified at two levels to work with circular layouts. First, the approach is now based on *angles* rather than *distances*. The displacement of vertices (labels) is also performed in circular fashion. Second, only the label vertices are allowed to move. The other vertices (anchors) are frozen and thus do not move during the layout configuration process.

A repulsive force is established between the labels. This force depends upon the amount of angle between them. For two labels v and u, α refers to interior angle formed by them with respect to circle center o. The repulsive force $f_{repulsion}$ is inversely proportional to α:

$$f_{repulsion}(\alpha) = -k^2/\alpha$$

where k is an ideal angle (e.g., 0.2 radians) between labels v and u. $f_{repulsion}$ is calculated between every pair of labels and summed.

An attractive force $f_{attraction}$ is established between the labels v and associated anchors a_v. Let position p_v be the radial projection of a_v in radial layout. The position p_v on the circular ring attracts the label v. β refers to the interior angle between v and p_v with respect to o.

$$f_{attraction}(\beta) = \beta^2/k$$

where k is a tolerable angle (e.g., 0.2 radians) between v and p_v. Note that, the above setting of attractive forces is for radial layout (as the label v is attracted towards p_v). For silhouette-based layout, p_v is set according to the orthogonal projection of anchor points on the silhouette boundary.

Figure 6.18 illustrates the force-directed annotation layout approach. Filled enumerated circles represent labels which are arranged on a circle, whereas tiny filled circles represent anchor points. Solid blue lines indicate repulsive forces and dashed red lines indicate attractive forces between labels and their ideal positions.

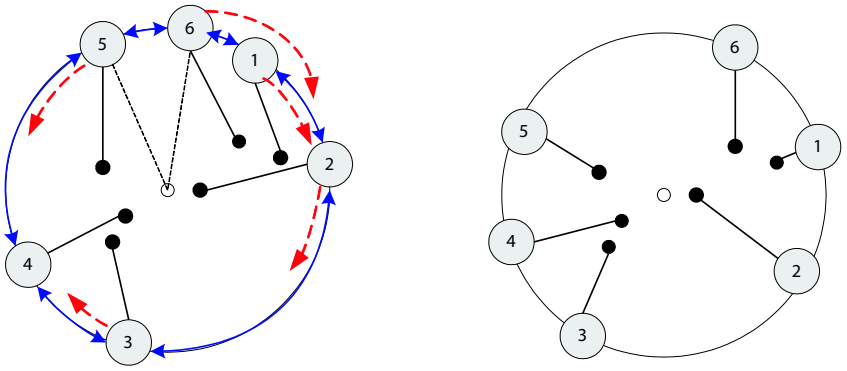


Figure 6.18.: Force-directed approach to spread labels around circular orbit. Before (*left*) and after configuration (*right*).

In order to determine the force-directed layout configuration, the repulsive and attractive forces are calculated for each label, and the label is displaced in an attempt to converge to an energy-min

Figure 6.19 shows the results of force-directed approach when applied on a radial layout. The radial layout determines the positions of labels using local information (anchor point and the center of circle around the graphical model) which may result in clustering of labels in a region and long connecting lines. On the other hand, a force-based approach performs the label placement in global context, by taking into account the positions of other labels in the layout. As a result it avoids cluttering of labels. The labels are pushed into the nearby free areas around the graphical model rather than being stacking on top of each other in an already packed region.

Orthogonal Layout

The algorithms described so far are able to produce annotation layout with straight connecting lines between anchor points and designated labels. In orthogonal layout style, connecting lines are axis-aligned lines that bend at orthogonal angles (see figure 6.20). A straight-line annotation layout, be it *flush* or *circular*, can be converted to orthogonal layout fairly easily:

1. Compute external annotation layout in any layout style,

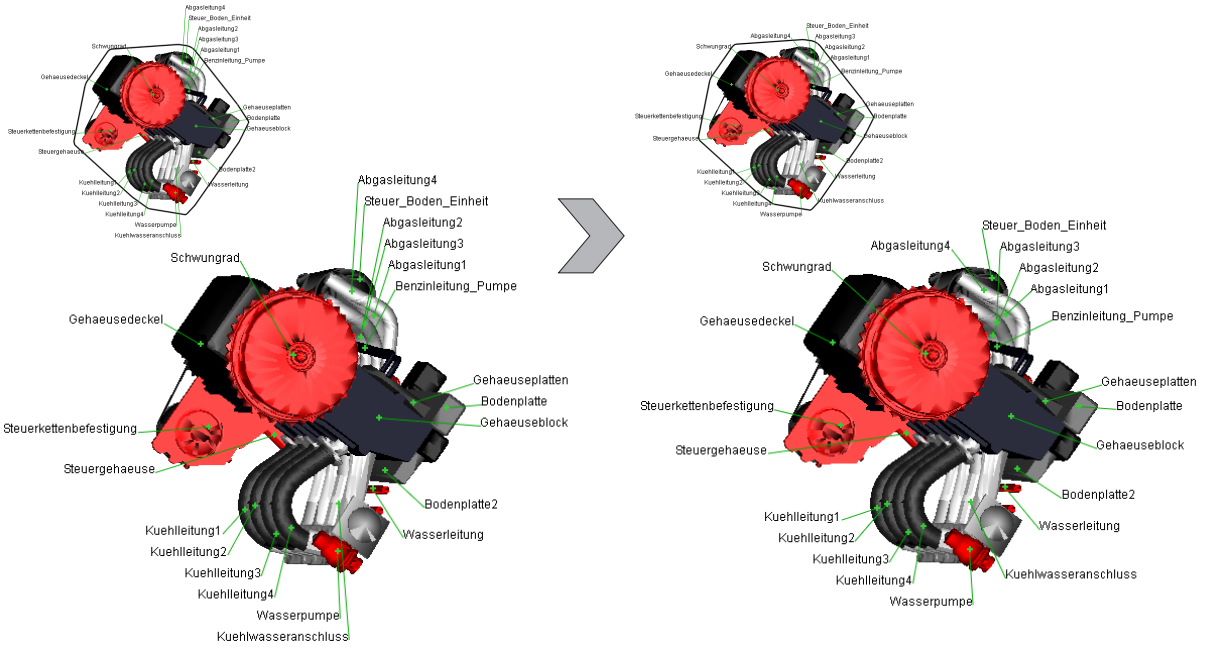


Figure 6.19.: Force-directed annotatation layout in circular style. *Left:* Radial layout (drawn with and without convex boundary). *Right:* After applying Force-directed configuration. Label placement close to the silhouette boundary of convex hull ultimately reduces the average length of connecting lines.

2. Draw the connecting lines in orthogonal style, and

3. Resolve intersections of orthogonal lines.

The orthogonal layout algorithm imposes two restrictions on the connecting lines: (i) only one bend is allowed (i.e., connecting lines can employ a vertical and a horizontal segment) and (ii) vertical segments connect anchor points and bends, while horizontal segments connect bends and labels.

In order to detect line intersections in the orthogonal layout, each horizontal segment is tested for intersection with all vertical segments. Every time an intersection is found, the label positions are exchanged. This procedure is continued until no intersections remain.

6.4.1.4. Layout Compaction

In order to keep the connecting lines short, this step aims at moving the labels towards their anchor points. Two different methods, based upon the kind of silhouette they use, were developed. The first method approximates the silhouette of graphical model by its convex hull. It computes intersections between connecting lines and the edges of the convex

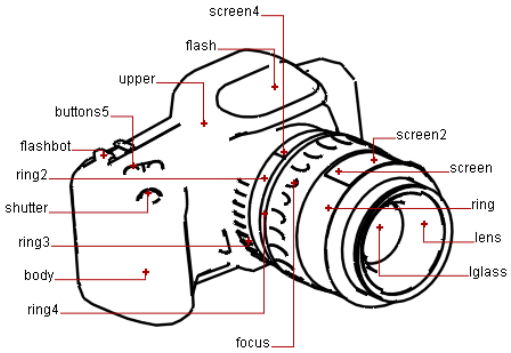


Figure 6.20.: Orthogonal layout in flush left-right style.

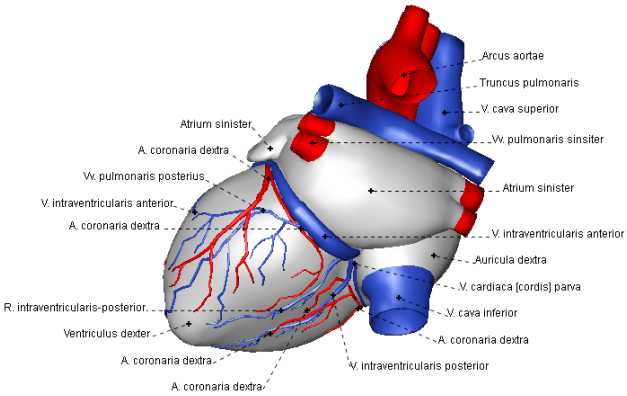


Figure 6.21.: Layout compaction. Initially labels were positioned using flush left-right layout and then moved inwards close to the actual silhouette pixels.

hull and re-targets the labels on the intersection points. The second compaction method uses the actual boundary pixels of objects in ID-buffer. After computing an annotation layout style, the labels on the left and right (or top and bottom) sides of the graphical model are slided towards the direction of anchor points until they hit the boundary pixels of the model. During this step, new line intersections may arise that are resolved once again. After layout compaction, the labels on both sides of the graphical model closely follow the actual boundary of the graphical model (see figure 6.21).

6.4.2. Internal Annotations

The strategy for internal placement of annotations is outlined in algorithm 8. The objects of interest in the ID-buffer are skeletonized and transformed into a graph (*skeleton graph construction*). Graph algorithms are used to analyze and select the best path from skeleton graphs (*path selection*) for *letter placement*. *Path smoothing* step ensures that the annotation path does not have steep curves and is smoothly curved.

Algorithm 8 Internal Annotation Layout Algorithm

1: `SkeletonGraph` Skeleton graph construction
2: `PathSelection:` Path selection
3: `Smoothing:` Path smoothing
4: `Placement:` Letter placement

Skeleton graph construction

For simple and efficient skeletonization, a scan-line algorithm [Peu81], is chosen that

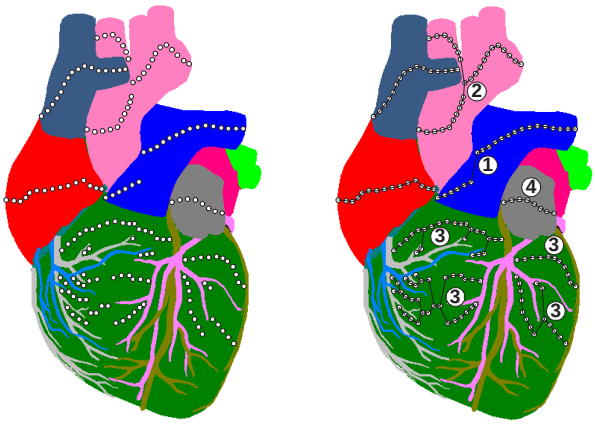


Figure 6.22.: Skeleton midpoints and skeleton graphs for an ID-buffer.

requires just a single pass on the 2D image data. Along the horizontal axis, the algorithm determines objects' spans in vertical direction. The horizontal axis was chosen because it produces horizontal aligned midpoints (see figure 6.22-a) which is the preferred reading direction. The costs of a second scan in horizontal direction outweighs the benefit in graph quality. Subsequent midpoints are connected and form a directed acyclic *skeleton graph* (see figure 6.22-b), specifying the predecessors and successors for each midpoint on the scan-line.

For graph traversal, a simple algorithm was developed to extract the stroke paths from the skeleton graph. Moreover, the available space for each stroke segment in y direction is also extracted, which is used to evaluate text stroke candidates.

All edges of the skeleton graph are assigned uniform weight. In the next step, connected components of the skeleton graph are extracted. Figure 6.23 displays the connected subgraphs of the skeleton graphs for some geometric objects in figure 6.22-b. Note, that there are four connected subgraphs for object 3.

Path selection

In this step the stroke candidates are evaluated in order to determine the most appropriate stroke path according to a set of labeling attributes (cf. section 6.2). Various functions are applied to measure and control the legibility and unambiguity of annotations.

A *steepness* function measures the steepness of stroke path with respect to horizontal axis. Since the horizontal lettering is preferred, the angle between the vector from the

start-point $v_1 = (x_1, y_1)$ and endpoint $v_n = (x_n, y_n)$ of a text stroke and the horizon is used as cost factor:

$$C_{steepness} = \arctan(\frac{\Delta y}{\Delta x})$$

A *curvature* function measures the curvature of the stroke. This information is used select the stroke paths with less complicated shapes and less curvature. The angle $\alpha(i)$ between adjacent vectors e_i and e_{i+1} in the text stroke is computed by their dot product. The cost function penalizes changing directions over a a path of length n: Complicated and extreme types of curvature in the stroke path are avoided to ease legibility.

$$\alpha(i) = dot(e_i, e_{i+1})$$

$$C_{curvature} = \frac{1}{n-1} \sum_{i=1}^{n-1} |\alpha(i) - \alpha(i+1)|$$

Area features have to provide enough space to contain internal labels, otherwise they are considered as point features and labeled externally. Therefore, the visible areas of the geometric objects and their extent are considered to decide the annotation type (internal or external). A *surrounding space* function computes the vertical span (*height*) of object at each vertex v_i for a given path (*Path*) through the skeleton graph:

$$C_{space} = \sum_{v_i \in Path} \text{height}(v_i)$$

It is not optimal to place text strokes over very narrow areas because at these positions the visual object might divide up into several parts. A *minimum space* function measures the location of smallest vertical span (*height*) of object in a given path (*Path*) through the skeleton graph:

$$C_{bottle-neck} = \min_{v_i \in Path} (\text{height}(v_i))$$

Often the text strokes of labels do not require the full path length. Also, the stroke path may consist of many branch segments connected together. A *path length* function evaluates the number of edges and their lengths within the path, which is used to select the segment within the chosen path which has the length greater than the length of internal annotation. Figure 6.23 displays the connected subgraphs of the skeleton graphs for some geometric objects in figure 6.22-b. A maximal path through these maximal subgraphs is

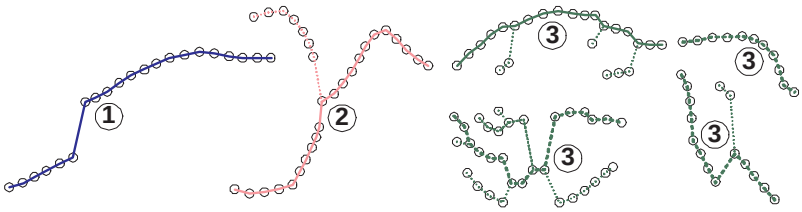


Figure 6.23.: Skeleton graph: longest paths are emphasized.

determined. They are emphasized by bold dashed line strokes in figure, whereas the rest of the candidate paths are indicated by dotted lines.

Path smoothing

After extracting the appropriate path from the skeleton graph, the chosen path is filtered in order to reduce the curvature and the impact of the discontinuities.

In cartography and computer graphics, line simplification algorithms are widely used to remove noisy, redundant and irrelevant vertices from poly-lines while preserving the perceptual properties of the original line. A well-known example is the elegant Douglas-Peucker [DP73] algorithm. In the proposed annotation framework, line simplification emphasizes outliers caused by the discontinuities of the run-length algorithm. Therefore a dynamic mean filter was chosen to smooth extreme changes in the chosen path (see dashed border in Fig. 6.24). The algorithm determines new positions v_i' for all vertices v_i according to a path segment with a variable width m:

$$v_i' = \frac{1}{2m+1} \sum_{j=0}^{m} v_{i+j}$$

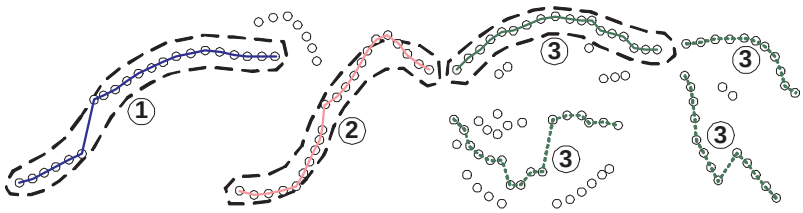


Figure 6.24.: Path smoothing with a mean filter.

Letter placement

When the available path length exceeds the required length to display the original or an abbreviated text, the path segments with low curvature are searched within salient regions. Figure 6.25 presents several alternatives to display the text stroke on the skeleton. To

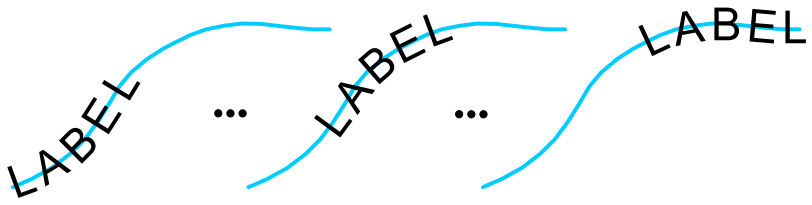


Figure 6.25.: Placements of text strokes on the skeleton.

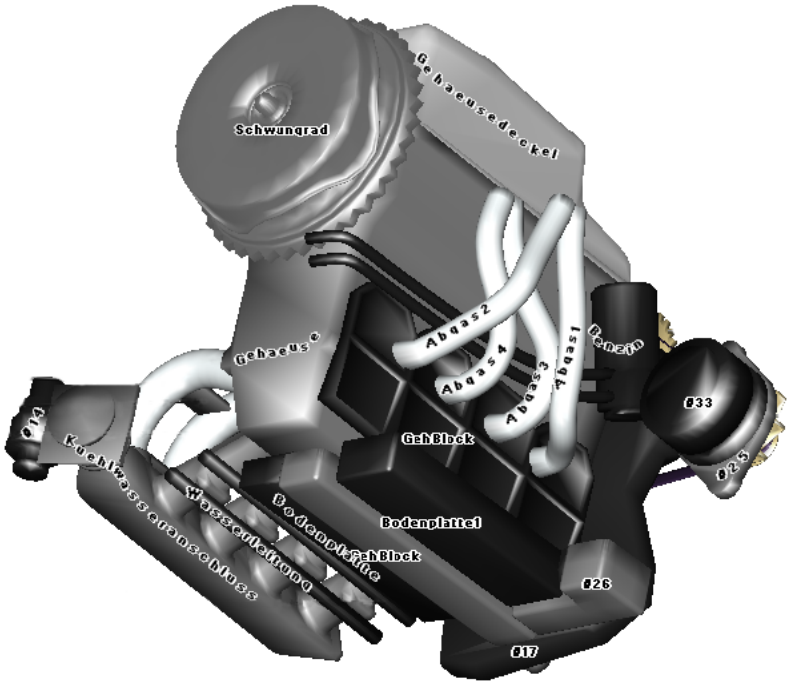


Figure 6.26.: Placement of internal annotations in the image.

select the best position of the text strokes, all stroke segments are evaluated with respect to the same metrics as applied to the skeleton paths. For further improvement of annotation path, additional post-processing step can be applied. For example, the curvature of the labels path can be decreased on the basis of user-defined parameters. Via specifying the number of sampling points (normally one per letter) the stiffness of the path is controlled. Thus, if using only two sampling points, a straight line path is obtained. Finally, the letters of internal labels are aligned on B-splines and thus rotated in coordination with the curve slope. Therefore, it is very important to choose a font which is robust enough to maintain a sufficient readability on low resolution screens. Simple fonts such as the font family of the *Serif-less Linear-Antiqua* are selected to render internal as well as external annotations. Figure 6.26 shows the result of placing internal annotations in the image.

6.4.3. Frame Coherency

So far the discussion of annotation placement covered only static frames—*to determine an annotation layout for an image*. The outcome of this process is, the positions of internal labels and external labels in accordance to the aesthetics and function attributes. Nevertheless, in dynamic environments, computing label layouts solely from individual frames would result in flickering and would distract the user. To reduce the amount of visual discontinuity in the presentation, the layout strategy is revised to take into account the outcomes from the previous frames so that the layout appears *stable* or frame coherent over the course of time.

External annotations

For external labels, four techniques are employed to achieve frame coherence: (i) anchor points are stabilized, (ii) the layout is stabilized, (iii) labels are animated from old to new positions, and (iv) animation techniques are chosen to displace the labels and appear/disappear them when needed. Figure 6.27 illustrates this approach in four layers.

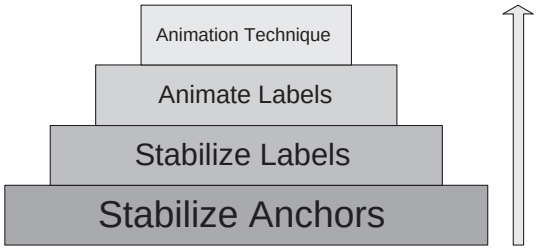


Figure 6.27.: Different stages of frame coherence for external annotations.

All external annotation layout style algorithms rely heavily on the positions of anchor points. Therefore, movements of anchor points might induce a global change in the layout. To enhance the frame coherency, a new heuristic for placing anchor points is added—If possible, retain previous positions of anchor point in the next frame. However, it can happen that, as a result of object movement, the anchor point no longer overlays its object, or its position on the object becomes very poor. In both cases, it's the time to shift the anchor point to new position.

In order to implement this new heuristics, an *attractive force* is added to the distance-buffer which aims at keeping anchor points close to their previous positions. For each anchor point c belonging to an object o at frame $t-1$, an *additive function* modifies pixels p of object o in distance-buffer D at frame t.

$$D_t(p) = D_t(p) + f(|p - c_{t-1}|) * k \quad \forall p \in o$$

where f is a non-negative decreasing function, $|p - c_{t-1}|$ is the distance between the pixel p in D_t and the corresponding anchor point c at time $t-1$; whereas k is a scaling factor. The result of this function is that the distance values in the next frame are incremented based on the anchor point locations from the previous frame, that increases the probability of old anchor positions being selected again as anchor points in the next frame.

Figure 6.28 illustrates this idea in 1D. The additive function creates high peaks on previous anchor positions and increases their probability for being selected as new anchor points. In the figure, the $A(t)$ and $B(t)$ refer to the anchor points of the green and red object in frame t. In the next frame $t+1$ the additive function (in blue dotted line) modifies D and creates new peaks. The global maxima $A(t)$ and $A(t+1)$ for the green object are identical, so that its anchor point remains stable. However, there is now a new global maximum $B(t+1)$ for the red object, so that its anchor point moves to another location. This illustrates how the old anchor positions are retained as long as they are acceptable, otherwise the anchor points jump to the better candidates.

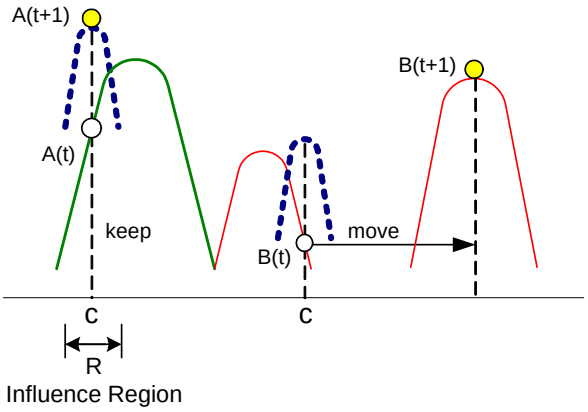


Figure 6.28.: Stabilizing anchor points.

After stabilizing anchor points, the next step is to avoid the labels from jumping from one side to another side in the layout. In this regard, the pivot point (cf. subsection 6.4.1.3) of flush layouts should also be stabilized because the assignment of labels to regions in flush layout takes place with respect the pivot point. However, if the pivot point remains steady for a fairly long time during the course of user interaction, the numbers of labels per region can get very unbalanced. Therefore, a pivot element is nailed as long as it provides an acceptable ratio (for example 2:1) of the number of anchor points on the two sides of the layout, otherwise it is set to new location. For circular layouts, the center point and the silhouette boundaries are updated in lazy manner so as to retain the old label positions.

When it is inevitable to update the layout and shift the labels to new position, additional steps are required to make sure that the transition goes smoothly. For that cause, different

animation techniques, such as *slow-in slow-out* motion [FvDFH90], are used to displace the labels and connecting lines. When the graphical objects become visible/invisible, the corresponding labels are *faded-in/faded-out* to avoid the sudden label appearance/disappearance. This gives the user a little more time to expect what changes are going to happen soon in the layout. Moreover, when the labels are being animated, their intensity is decreased to reduce the total amount of intensity moved in the layout.

Internal annotations

As a result of change of point of view of graphical model, even minor changes in the shape of objects may result in drastic changes of the skeleton, that would eventually lead to the change in the position and shape of the internal annotation. Therefore an additional measure is added to evaluate the similarities of text strokes in subsequent frames, using *squared distance* function. For a frame at time t, sum of squared differences of each letter's position c_{pos} in the string *Char* is calculated from that letter's position in the frame $t-1$:

$$C_{distance} = \sum_{c \in Char} (|c_{pos(t)} - c_{pos(t-1)}|)^2$$

By increasing the weight of this function, the labels get stickier and the lettering becomes more frame coherent during user interactions.

When labeling objects, internal annotations are preferably chosen for objects that have large free area around the object skeleton (cf. subsection 6.4.2). The internal labels within the large objects do not jump between alternative positions as often as in small objects. Thus, this metric reflects the probability of a frame-coherent label placement.

As done for external annotations, the positions and orientations of the text letters are also animated between subsequent frames.

6.4.4. Adapting Annotation Layout Type

During the user interaction, the annotation layout for the underlying visualization needs to be updated in real-time. As a result of zooming, the style of annotations might also need to be adapted to the current view of the graphical model. The annotation module determines the best way to annotate the illustration. Due to the spatial continuity principle [May01] that recommends placing related items as close as possible, internal

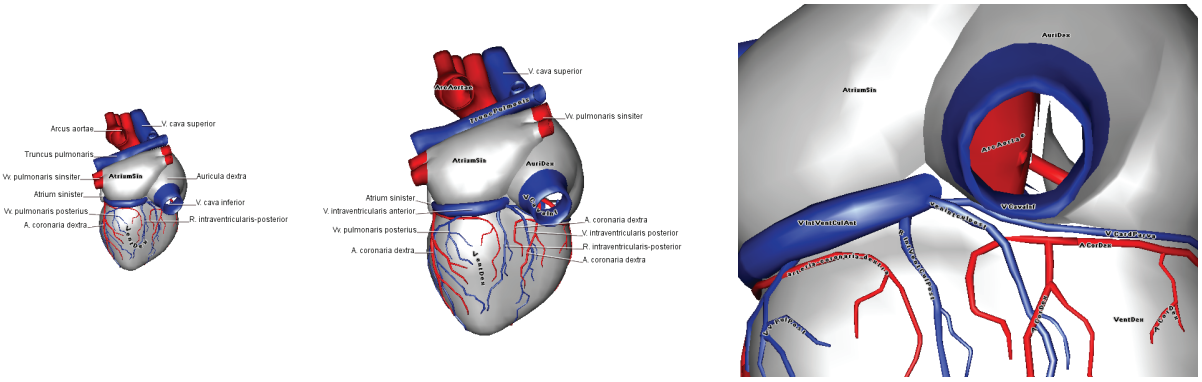


Figure 6.29.: Adapting annotation type in the layout when the viewing context changes.

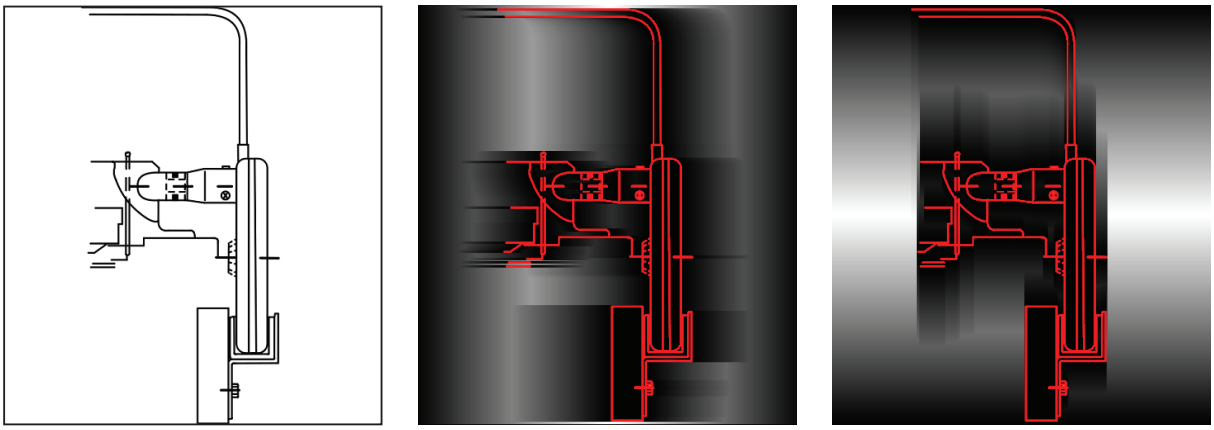
annotation are preferred over external annotations. If there is not sufficient space to annotate an object internally (even using the abbreviated text) an external placement is chosen for that particular object. In order to ensure legibility of text, an external annotation is chosen for an object instead of internal when stroke path of an internal annotation is poor, e.g., presence of steep curvature. Conversely, when there is a little or no room at background space to place the external annotations, the system resorts to the internal placement of annotations. An illustration can contain arbitrary number of internal and external labels at the same time. In order to reduce the effect of optical flow of elements, such adaptations should be minimized.

Figure 6.29 shows three frames from a zooming sequence to exemplify the dynamic label adaptation.

6.5. Extensions & Applications

Space-Efficient Labeling

The annotation layout methods described so far provide solutions for a variety of labeling problems, but the layouts are not optimized to PDA viewing devices. The restricted display size of mobile device requires that the annotations must be strictly placed within the given space, without compromising the layout quality too much, and increasing the number of annotations in the layout. Thus, a novel *space-efficient* layout algorithm was developed which uses distance transforms (cf. section 6.3.1) not only to compute anchor points and internal labels, but also to locate the empty regions in the negative space.



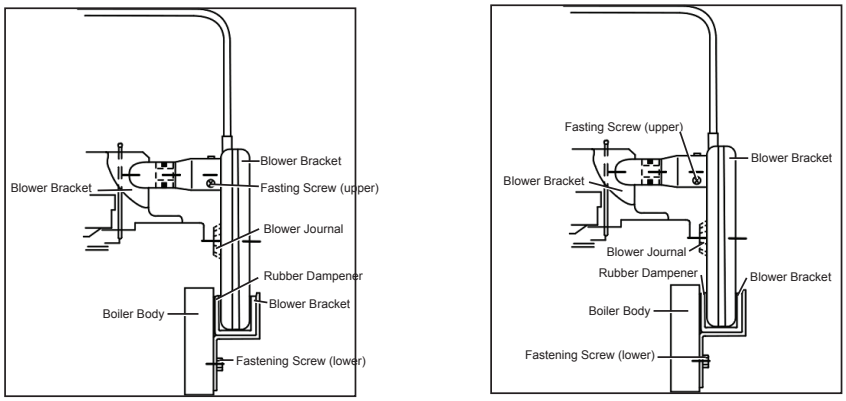
(a) Source illustration (b) Distance-buffer (x-direction) (c) Distance-buffer (y-direction)

Figure 6.30.: A source image and the corresponding distance-buffer X and distance-buffer Y.

The standard distance transformation algorithm is modified in order to get the minimal distance from the boundary pixels in x and y direction. Two distance buffers, namely distance-buffer X (D_x) and distance-buffer Y (D_y), are computed (cf. figure 6.30-b and figure 6.30-c). The label placement is performed in a greedy fashion with the most relevant object annotated first. For each object, the system analyzes distance buffers D_x and D_y to find a region R of width w and height h nearest to the anchor point, and capable to accommodate the annotation. The annotation is rendered in the final layout as well as in the ID-buffer. The key here is to refresh the ID-buffer and distance-buffer every time after adding an annotation in the layout. The space-efficient procedure is outline in the following:

1. select the most relevant object according to task model or user interaction,
2. find R of size (w, h) in D_x, D_y nearest to the anchor point,
3. place the annotation in R in layout,
4. render the annotation in the ID-buffer,
5. update distance-buffers D_x, D_y from ID-buffer,
6. repeat.

The results of space-efficient labeling (cf. figure 6.31-left) could be compared to the layout done using *flush left-right layout* style. The previous approach (cf. figure 6.31-right) relied on first placing the labeling strictly on the left and right sides of the graphical object and



(a) Flush left-right layout (b) Space-efficient labeling

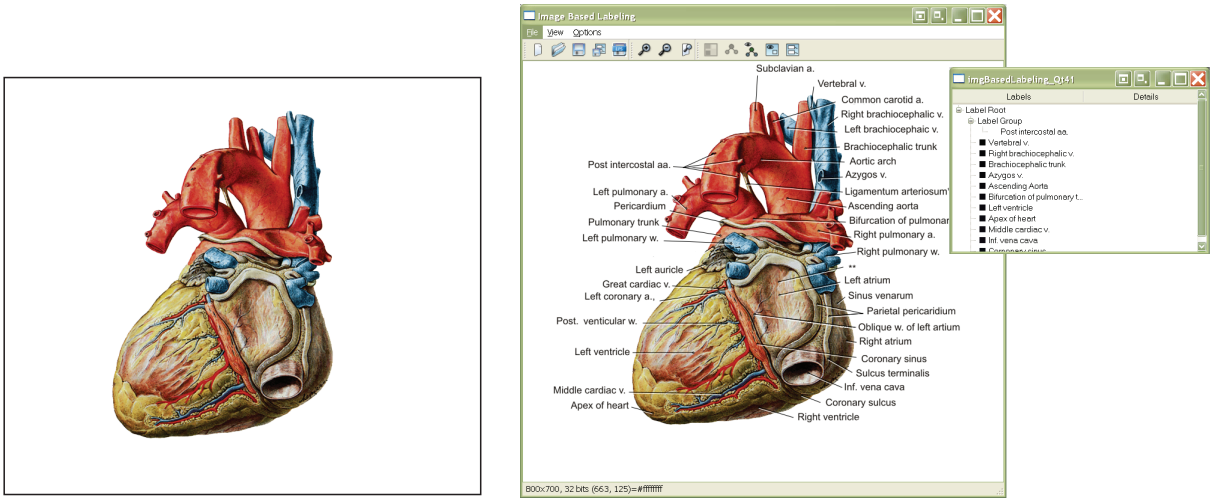
Figure 6.31.: Results of *flush left-right* layout as compared to space-efficient labeling.

(a) Original image (b) ID-buffer (c) Annotated image

Figure 6.32.: Space-efficient labeling on a bicycle image.

then bringing them close to the silhouette boundaries without taking into account the space that is already available on the image. The new approach explicitly looks for the nearest locations to the objects where the corresponding labels could be placed and thus makes efficient use of existing space without increasing the illustration size.

The algorithm uses free space very efficiently even if the space is divided in small and irregular chunks (see figure 6.32). A drawback of the approach is that label-line overlaps may occur in the layout. If the layout contains non-uniformly sized labels, intersections among connecting lines may not be resolved. This is due to the fact that the positions of two labels of different size cannot be swapped without the risk of overlapping other objects/labels in the layout.



(a) Source image (b) Image-based labeling editor

Figure 6.33.: Authoring illustrations in image-based labeling editor. *a:* A new labeling project is created and source image is loaded. *b:* Automated label placement in image-based labeling editor. The author picks anchor points locations in the image using mouse and types text labels for respective anchor points. The annotation layout is set automatically by the system.

Image-Based Labeling Editor

Since, an annotation layout is sensitive to the context and viewing scenarios, often illustrations need customization before they can be integrated to documents. An image-based labeling editor was built to create annotation layouts for 2D illustrations. Authors are required to provide content and identify regions that should be annotated. The system does the rest of work to create the annotation layout which is based on user preferences and viewing requirements. An author loads a 2D image in the image-based labeling editor and types the labels. The anchor points are extracted automatically from the ID-buffer. If ID-buffer is not provided, the author specifies anchor locations by clicking mouse on the image, and indicates the color of background space where the external labels would be placed. Using the automatic/manual anchor points and text labels, the annotation module computes an optimal annotation layout for the image (see figure 6.33). In addition, authors can specify user preferences, such as layout styles, fonts or alternative descriptions. The annotation project file can be saved for future use and shared with other users. External applications can use a annotation project to send own requests to the annotation module. The image-based labeling can also export annotated illustrations in various common formats such as PostScript, PDF, PNG and XML.

Interfaces

A set of solutions were developed to integrate the annotation layout module to external applications including image-based labeling editor:

- *Stand alone 3D application:* A stand-alone application was built to demonstrate the external annotation layout styles given in this chapter [AHFS08, AHS05]. The application also integrates legends and descriptions in the layout. The second application was a collaborative work [GAHS05a] which unified internal and external annotation styles in a single framework. In both applications, the users can explore 3D polygonal models of their choice while the system computes annotation layout dynamically.

- *Shell-based application:* A shell-based interface (`labeler.EXE`) was developed to give access to the annotation layout module. An external application makes call to `labeler.EXE` and provides the command line parameters. The command line parameters includes path to XML file which contains the annotation table (see subsection 6.3.2) and the output format. Based on the input parameters, the `labeler.EXE` returns the final annotation layout either in XML format or a transparent layer with the annotation rendered opaque on it which can overlaid on the source illustration. Shell-based annotation module can be deployed in a client-server architecture, where the application runs the server. A client makes calls to the server and the server process the request and generates the response. This is especially useful when running full-fledged labeling algorithm on mobile devices is likely to pose a serious performance bottleneck. Using a server-side annotation module the complex tasks are shifted to the external server and the client integrates the results.

- *Mask and task editor:* An example of server-side annotation layout module is demonstrated in Task and Mask editor in figure 6.34 [FLH+06]. Task and Mask editor generates adaptive visualizations in a mobile maintenance support scenario by making calls to the annotation layout server. The visualizations in the system are backed by the task model 4.4. The task model determines the relevance of components within the steps of a maintenance operation and filters the information that need to be displayed on the small screen. The annotation layout is requested to create annotation for only relevant components.

- *Web browser plugin:* To enhance the web pages with dynamic illustrations, a pluggable version of annotation layout module was developed which can be embedded into a web browser. Contextual parameters set by the host application are taken into account when setting the annotation layout.

- *API:* A powerful, fully-functional object-oriented API has been developed which consists of a collection of classes and member calls giving access to the core features of annotation layout module library. The host application, such as adaptive document framework, uses the API to integrate interactive illustrations to the documents [AHFS08]. Another example of host application which uses the annotation layout module API is SearchIllustrator [GGA+07, GGA+06].

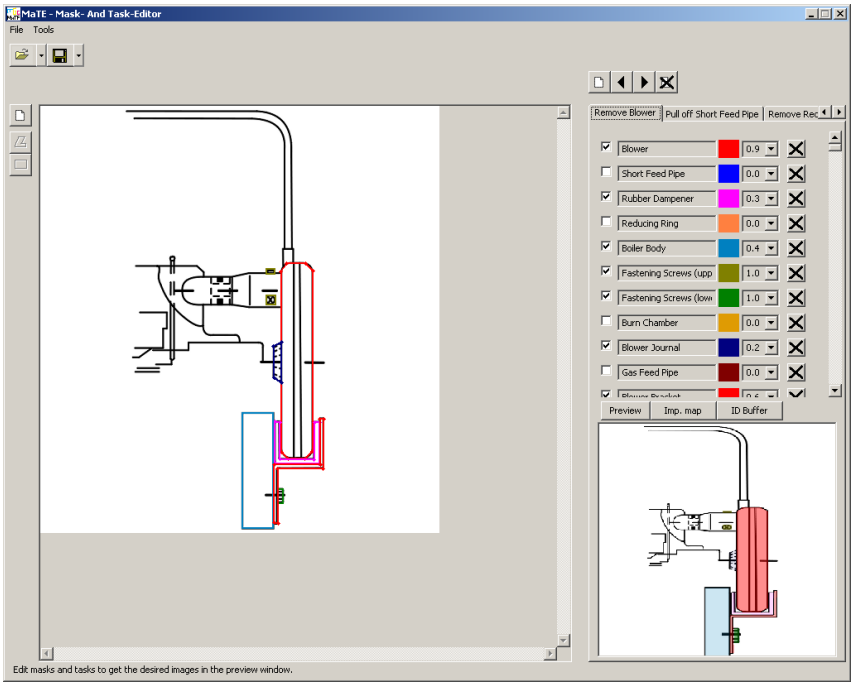


Figure 6.34.: Task and Mask editor.

6.6. Summary

This chapter proposed an automated annotation layout framework for annotating illustrations in adaptive multimedia documents. The annotation layout problem is dealt by developing heuristics those take into account various aesthetics and functional attributes from the hand-made illustrations and interactive media. One of the novel aspects of this

work is the variety of layout styles that can be produced by the framework. The annotation approach presented in this chapter works on 2D projections of the graphical model, it is equally suitable to annotate 2D illustrations as well as 3D visualizations. While the major part of algorithmic section was devoted to the external labeling, further details about the internal annotation placement can be found in [GHS05, GHS06].

An initial numerical evaluation was conducted [Ali05] to evaluate the goodness of automated annotation layout. The numerical evaluations were based on various functional and aesthetic attributes (cf. section 6.1). The attributes were translated to metrics—such as number of label overlaps in the layout, number of line intersections, average length of connecting lines, average displacement of labels over the time, etc. [HGAS05]—which can be evaluated dynamically by the system. The results from the early experiments [Ali05] were very positive and gave us a good insight towards improving the algorithms. Unfortunately, there is no benchmark data to compare the results with other systems. Indeed, an empirical evaluation, based on user study, is needed to further evaluate the effectiveness of automated labeling.

The annotation layout techniques presented in this chapter were the pioneering work in the field of adaptive illustrations. Our work has been gaining the attention of research community[2] and has influenced further research in the field of automated annotation layouts.

2 32 citations of [AHS05], 13 citations of [HGAS05], 12 citations of [HAS04], etc. Stats by Google Scholar as of March 25, 2009.

Chapter 7

Conclusion and Future Work

This dissertation explored the topic of automated layout adaptation of digital documents having an abundance of illustrations. A novel approach is presented that combines constraint-based optimization with physical forces to modify the document layout. In particular, the approach also incorporates unique annotation layout methods that adapt the embedded illustrations to match the contextual requirements over the time. In this chapter, we review the contributions of this work and suggest potential areas of future research.

7.1. Concluding Remarks

Effective presentation of rich and interactive illustrated material on a variety of display devices raises many design issues with regard to authoring and adaptation of content/style. On the one hand, it requires that the information content and the design solution be tailored to individual requirements of the mass audience, which would ultimately put tremendous burden on the authors. On the other hand, the dynamics of digital media may restrict or prevent the authors to foresee all the possible communication scenarios. If no particular care is taken in designing the content and style for digital documents, the reading experience would be inconsistent from one screen size to another. The problem is not only limited to the overall appearance of the document page, but also affects the annotated illustrations within the page. The readability of illustrations, if they are of static nature, may suffer when displayed at size other than the intended resolution. Recognizing this

problem, this thesis examines a framework for automated layout adaptation of documents as well as dynamic illustrations. The solutions and the examples herein illustrate the potential application of the framework in various practical scenarios. However, a thorough and extensive study is needed to test the validity of the framework by many users over long time. Nonetheless, on the basis of early results, the framework demonstrates a capacity to contribute to the improvement of dynamic design solutions in interactive environments.

This research especially focuses on design problems associated with the rich illustrative material in digital realm. A generic framework has been developed [AHFS08] which provides comprehensive support for overall adaptation of digital documents in continuous manner. For more specific problems in digital documents, such as adaptation of annotations in illustrations, the framework presents a set of novel solutions which are specialized to the annotation tasks in interactive 2D/3D illustrations, but other domains such as information visualization may also benefit from them.

The generic adaptive document layout framework resembles the general document design process—content, layout, and interaction—where different roles are accomplished by respective users, namely authors, designers and viewers. In this process, content authors prepare the content representation at multiple granularity levels that can be used to create different versions of documents. Using a novel task-driven approach, content authors can specify different views of underlying content and indicate the relevance of particular objects in illustrations with respect to the task at hand. The role of designer is to provide template design rules to create adaptable design for a range of viewing scenarios (e.g., different screen aspect ratios or device classes). Finally, on the client side, the viewers request information and have the ability to interact with documents.

The final look of an adaptive document is determined by the combined efforts of constraint-based optimization and application of physical forces on selective regions of the layout, thereby, the forces expand the range of possible designs that can be generated from the given template. Force-directed algorithms in this thesis provide the effective means to achieve an optimal placement of the content when the layout templates are partially designed or even absent. The way of computing dynamic solutions can make the authoring process efficient by suggesting alternative design solutions to expert designers. During run-time, a dynamic solution would allow the viewers to customize the presentation according to their own preferences. Since the layout is a crucial part of any presentation system, many application areas—WWW, multi-display documents, mobile e-manuals, and interactive visual encyclopedias to name a few—can benefit from this research.

The annotation layout component in this thesis provides significant improvements over our previous work in [HAS04] which only dealt with external annotations in 2D images. The improved approach deals with both internal and external label placement and is equally suitable to annotate 2D illustrations and 3D interactive visualizations in the real-time. As optimal annotation layout is an NP-hard problem, the annotation layout approach is heuristic-based which takes into account various aesthetics and functional attributes from hand-made illustrations and interactive media [HGAS05, GAHS05b]. In this regard, a variety of annotation layout styles [AHS05] were developed which are inspired from anatomical and technical illustrations drawn by expert designers.

Interactive annotated illustrations in our system provide many advantages over their static counterparts, such as, the users may query objects by annotations and vice versa, switch on/off annotations, choose level-of-detail of objects for annotating them, select alternative versions of content in labels and stylistic preferences, etc. For 3D illustrations, the adaptive annotations [GAHS05a] would guarantee a valid frame coherent layout throughout the course of user interaction. A set of solutions, such as stand-alone applications, image-based labeling editor, shell-based interface in client-server architecture, space efficient labeling for mobile devices [FLH+06], and full-fledged API, were developed to provide various features of annotation layout component to the external world.

It is worthwhile to note that the automated annotation solutions in this research serve to carry out the annotation tasks in the realm of dynamic media where it is impossible for the human illustrator to predetermine annotation layout for any imaginable context. Indeed, a user study needs to be conducted to evaluate the effectiveness of the automated solution. Potential areas of application of the proposed annotation layout component are in 2D and 3D visualizations, text illustration [GGA+07, GGA+06], dynamic maps, 3D visualization, augmented/virtual reality, interactive visual dictionaries, information visualization, etc. In traditional media, our image-based labeling editor provides convenient ways to create annotation layouts for existing 2D images.

The success of dynamic documents not only relies upon automated content/style adaptation, but it also rests on how authors design the content and style. Authoring of comprehensive content and style lay an ultimate foundation for the future adaptation of presentation over time. Albeit the main theme of this research was the adaptation, a set of tools essential for authoring adaptive documents were also presented. We conclude with the statement that by combining the authoring process—multi-representation content, template designing, and task models—with a variety of layout tools—automated content/template selection,

integrated force-driven and constraint-based layouts, and adaptive annotated illustrations—we can develop a powerful system that can effectively describe a range of dynamic design solutions for digital documents.

7.2. Future Work

A number of research directions can be developed from hereon based on the adaptive document layout framework.

The adaptive document framework relies on manual authoring of alternate content. Manual effort is essential for creating a new content, but we need an efficient way to transfer the existing content to the adaptive media. Research in the field of information retrieval, such as content extraction [GKG+05] from HTML documents and content characterization [KSP97] from scanned documents, can be exploited in this regard. Moreover, we can explore automated text summarization methods [MM99] to filter the important information from the source content to produce different versions of content. For the image content, adaptive document framework can exploit adaptive image cropping [CCGS07]. Adaptive image cropping aims to find an image-crop that suits the output display region in such a way that the key features of the original image are retained after cropping. Adaptive cropping would generate different versions of image content automatically instead of the layout engine depending on the designer to provide a set of cropped images for different design solutions.

Currently, the template authoring tool requires the designers to specify a set of *low-level* constraints, such as `left-of`, `top-align`, `width<200`, etc. . For a complex design, specifying large number of low-level constraints manually would be a tedious task. Hence, it would be useful to encode a collection of *high-level* design qualities [HNJ+04], such as alignment, page balance, regularity, compactness, etc. , that could be numerically translated to low-level constraints/forces and directly fed to the system. By using such high-level constraints, the designer would be able to quickly describe the design properties that must be satisfied in the final solution. High-level constraints would also make it easy to design the templates using a plain text editor. It would also be interesting to explore the modeling of high-level design qualities into physical forces within a force-driven layout.

Another interesting question is to determine a way of simultaneous editing of multiple templates in a layout style.

We can also explore automatic extraction of layout [MEA+03] from scanned documents. This would ease the time-consuming process of manual authoring of templates and let the system infer spatial constraints from the graphical examples.

The current implementation of adaptive document framework described in this work is a standalone client application. The framework can also be implemented as a server-side application. In this scenario, the adaptive document server would generate a personalized HTML page that can be displayed on any web browser installed at the client machine. However, because of limited layout support in the current web browsers, the translation of complex layouts to HTML page may be challenging.

In addition to 2D illustration and 3D polygonal visualizations, applications like medical tutoring systems also benefit from voxel-based volumetric visualizations. As the annotation layout approach proposed in this research essentially works on 2D projections of the visualizations, the annotation algorithms can be easily applied to annotate voxel visualizations. Another direction to explore is the direct use of 3D volumetric data, instead of 2D projections, to generate 3D distance-buffers and determine label placement in 3D space.

Another interesting project is automatic extractions of annotations (text, anchor points, connecting lines) from scanned illustrations using OCR (Optical Character Recognition) and straight line detection algorithms [LKJ06] to enhance the authoring process in image-based labeling editor. In addition, the image-based labeling editor may benefit from tools like "GrabCut" [RKB04] to perform semi-automatic foreground/background extraction in 2D illustrations.

The adaptive document layout framework should be tested with many design problems. An evaluation of the framework would give a deeper understanding to make necessary improvements in the authoring tool as well as dynamic design process. Furthermore, an independent user study is necessary to evaluate the impact of adaptive annotations in the learning process when compared to the traditional illustrations.

Bibliography

[Ad88] C. Arcelli and G. S. di Baja. Finding Local Maxima in a Pseudo-Euclidean Distance Transform. *Computer Vision, Graphics, and Image Processing*, 43(3):361–367, 1988.

[AEM01] O. Altamura, F. Esposito, and D. Malerba. Transforming Paper Documents into XML Format with Wisdom. *International Journal of Document Analysis and Recognition*, 4:2001, 2001.

[AF03] R. Azuma and C. Furmanski. Evaluating Label Placement for Augmented Reality View Management. In *IEEE and ACM Int. Symp. on Mixed and Augmented Reality*, pages 66–75, 2003.

[AHFS08] K. Ali, K. Hartmann, G. Fuchs, and H. Schumann. Adaptive Layout for Interactive Documents. In *8th Int. Symp. on Smart Graphics*, pages 24–35, 2008.

[AHS05] K. Ali, K. Hartmann, and Th. Strothotte. Label Layout for Interactive 3D Illustrations. *Journal of the WSCG*, 13:1–8, 2005.

[Ali05] K. Ali. Automated Realtime Label Layout for 3D Illustrations. Master's thesis, Department of Simulation and Graphics, Otto-von-Guericke University of Magdeburg, January 2005.

[APH+03] M. Agrawala, D. Phan, J. Heiser, J. Haymaker, J. Klingner, P. Hanrahan, and B. Tversky. Designing Effective Step-By-Step Assembly Instructions. In *30th Int. Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 828–837, 2003.

[AS94] C. Ahlberg and B. Shneiderman. Visual Information Seeking: Tight Coupling of Dynamic Query Filters with Starfield Displays. In *Proceedings of the*

SIGCHI Conference on Human Factors in Computing Systems: Celebrating Interdependence, pages 313–317. ACM Press, New York, 1994.

[AS01] M. Agrawala and C. Stolte. Rendering Effective Route Maps: Improving Usability Through Generalization. In *28th Ann. Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 241–250, 2001.

[Bai04] D. G. Bailey. An Efficient Euclidean Distance Transform. In *Combinatorial Image Analysis: 10th International Workshop, IWCIA*, volume 3322 of *Lecture Notes in Computer Science*, pages 394–408, Auckland, New Zealand, December 1–3 2004. Springer-Verlag GmbH.

[BBS01] G. J. Badros, A. Borning, and P. J. Stuckey. The Cassowary Linear Arithmetic Constraint Solving Algorithm. *ACM Transactions on Computer-Human Interaction*, 8(4):267–306, 2001.

[BDL00] C. Brun, M. Dymetman, and V. Lux. Document Structure and Multilingual Authoring. In *INLG '00: Proceedings of the first international conference on Natural language generation*, pages 24–31, Morristown, NJ, USA, 2000. Association for Computational Linguistics.

[Bea85] R. J. Beach. *Setting tables and illustrations with style*. PhD thesis, Dept. of Computer science, University of Waterloo, Ontario, Canada, 1985.

[Ber83] J. Bertin. *Semiology of Graphics*. University of Wisconsin Press, 1983.

[BF00] B. Bell and S. Feiner. Dynamic Space Management for User Interfaces. In *Symp. on User Interface Software and Technology*, pages 238–248, 2000.

[BFH01] B. Bell, S. Feiner, and T. Höllerer. View Management for Virtual and Augmented Reality. In *Symp. on User Interface Software and Technology*, pages 101–110, 2001.

[BFH02] B. Bell, S. Feiner, and T. Höllerer. Information at a Glance. *IEEE Computer Graphics and Applications*, 22(4):6–9, July/August 2002.

[BGMP01] O. Buyukkokten, H. Garcia-Molina, and A. Paepcke. Accordion Summarization for End-Game Browsing on PDAs and Cellular Phones. In *CHI '01: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 213–220, New York, NY, USA, 2001. ACM.

[BH95] B. B. Bederson and J. D. Hollan. Pad++: A Zoomable Graphical Interface System. In *CHI '95: Conference companion on Human factors in computing systems*, pages 23–24, New York, NY, USA, 1995. ACM.

[BHS+96] B. B. Bederson, J. D. Hollan, J. Stewart, D. Rogers, A. Druin, D. Vick, L. Ring, E. Grose, and C. Forsythe. A Zooming Web Browser. In *9th Annual ACM Symposium on User-Interface Software and Technology*, 1996.

[Bit] Bitstream. Bitstream - ThunderHawk. `http://www.bitstream.com/wireless/`.

[BKKW95] A. Brüggemann-Klein, R. Klein, and S. Wohlfeil. Pagination Reconsidered. *Electronic Publishing-Origination, Dissemination, and Design*, 8(2/3):139–152, June/September 1995.

[BKKW98] A. Brüggemann-Klein, R. Klein, and S. Wohlfeil. On the pagination of complex documents. Technical Report 234, Department of Computer Science, FernUniversität Hagen, 1998.

[BKLP01] D. A. Bowman, E. Kruijff, J. J. LaViola, and I. Poupyrev. An Introduction to 3-D User Interface Design. *Presence: Teleoper. Virtual Environ.*, 10(1):96–108, 2001.

[BKLP04] D. A. Bowman, E. Kruijff, J. J. LaViola, and I. Poupyrev. *3D User Interfaces: Theory and Practice*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2004.

[BKSW04] M. A. Bekos, M. Kaufmann, A. Symvonis, and A. Wolff. Boundary Labeling: Models and Efficient Algorithms for Rectangular Maps. In *12th Int. Symp. on Graph Drawing*, pages 49–59, 2004.

[Bor86] G. Borgefors. Distance Transformations in Digital Images. *Computer Vision, Graphics and Image Processing*, 34(3):344–371, 1986.

[BPGN04] E. Bier, K. Popat, L. Good, and A. Newberger. Zoomable user interface for in-depth reading. In *JCDL '04: Proceedings of the 4th ACM/IEEE-CS joint conference on Digital libraries*, pages 424–424, New York, NY, USA, 2004. ACM.

[Bra00] BrainInfo. *http://braininfo.rprc.washington.edu* 11.04.2004, 2000.

[Bri96] M. H. Briscoe. *Preparing Scientific Illustrations: A Guide to Better Posters, Presentations and Publications*. Springer-Verlag, 2nd edition, 1996.

[BS97] T. W. Bickmore and B. N. Schilit. Digestor: Device-Independent Access to the World Wide Web. In *Proceedings of Wide Web Conference (WWW6)*, pages 655–663, April 1997.

[BSM04] K. Berkner, E. L. Schwartz, and C. Marle. SmartNails - Image and Display Dependent Thumbnails. In *Document Recognition and Retrieval XI*, pages 53–65. SPIE, 2004.

[BTM+01] G. J Badros, J. J. Tirtowidjojo, K. Marriott, B. Meyer, W. Portnoy, and A. Borning. A Constraint Extension to Scalable Vector Graphics. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 489–498, New York, NY, USA, 2001. ACM.

[Buc91] M. K. Buckland. Information as Thing. *Journal of the American Society for Information Science*, 42(5):351–360, January 1991.

[Buc97] M. K. Buckland. What is a "Document"? *Journal of the American Society for Information Science*, 48(9):804–809, September 1997.

[Buc98] M. K. Buckland. What is a "Digital Document"? *Document Numérique*, 2(2):221–230, 1998.

[CCGS07] G. Ciocca, C. Cusano, F. Gasparini, and R. Schettini. Self-Adaptive Image Cropping for Small Displays. *IEEE Transactions on Consumer Electronics*, 53(4):1622–1627, November 2007.

[Chi82] *The Chicago Manual of Style*. University of Chicago Press, thirteenth edition, 1982.

[CJ90] A. C. Cook and C. B. Jones. A Prolog Rule-based System for Sartographic Name Placement. *Computer Graphics Forum*, 9(2):109–126, 1990.

[CM84] W .S. Cleveland and R. McGill. Graphical Perception: Theory, Experimentation, and Application to the Development of Graphical Methods. *Journal of the American Statistical Association*, 79(387):531–554, September 1984.

[CMS95] J. Christensen, J. Marks, and S. Shieber. An Empirical Study of Algorithms for Point-Feature Label Placement. *ACM Transactions on Graphics*, 14(3):203–232, 1995.

[col] Personalization: Collaborative filtering vs prediction based on benefit theory. `http://myshoppal.typepad.com/blog/2007/11/personalization.html`.

[CP86] J. M. Clark and A. Paivio. Dual Coding Theory and Education. *Educational Psychology Review*, 3(3):149–210, 1986.

[CSRS03] W. Chigona, H. Sonnet, F. Ritter, and Th. Strothotte. Shadows with a Message. In *3rd Int. Symp. on Smart Graphics*, pages 91–101, 2003.

[CYWM03a] D. Cai, S. Yu, J.-R. Wen, and W.-Y. Ma. Extracting Content Structure for Web Pages based on Visual Representation. In *Proceedings of Fifth Asia Pacific Web Conference (APWeb2003)*, 2003.

[CYWM03b] D. Cai, S. Yu, J.-R. Wen, and W.-Y. Ma. VIPS: a Vision-based Page Segmentation Algorithm. Technical Report MSR-TR-2003-79, Microsoft Technical Report, 2003.

[DAK+06] A. Dengel, S. Agne, B. Klein, A. Ebert, and M. Deller. Human-Centered Interaction with Documents. In *HCM '06: Proceedings of the 1st ACM International Workshop on Human-Centered Multimedia*, pages 35–44, New York, NY, USA, 2006. ACM.

[Dan78] P. E. Danielsson. A New Shape Factor. *Computer Graphics Image Processing*, 7(2):292–299, April 1978.

[DD06] M. M. Deza and E. Deza. *Dictionary of Distances*. Elsevier Science, 2006.

[DETT99] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, Upper Saddle River, NJ, 1999.

[DK97] M. C. Dyson and G. J. Kipping. The Legibility of Screen Formats: Are Three Columns Better Than One? *Computers & Graphics*, 21(6):703–712, 1997.

[DLR00] M. Dymetman, V. Lux, and A. Ranta. XML and multilingual document authoring: convergent trends. In *Proceedings of the 18th conference on Computational linguistics*, pages 243–249, Morristown, NJ, USA, 2000. Association for Computational Linguistics.

[Doc02] P. Docherty, editor. *DK Ultimate Visual Dictionary*. Dorling Kindersley Publishing, New York, revised edition, 2002.

[DOU93] C. Duursma, O. Olsson, and Sundin U. Task Model Definition and Task Analysis Process, 1993. ESPRIT Project P5248 KADS-II/M5/VUB/RR/004/2.0.

[DP73] D. H. Douglas and T. K. Peucker. Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or Its Caricature. *Canadian Cartographer*, 10(2):112–122, December 1973.

[DPP03] D. Dörschlag, I. Petzold, and L. Plümer. Automatic Labelling of Areas in Thematic Maps. In *21st Int. Cartographic Conf.*, 2003.

[DWE02] J. Diepstraten, D. Weiskopf, and T. Ertl. Transparency in Interactive Technical Illustrations. *Computer Graphics Forum*, 21(3):317–325, September 2002.

[DWE03] J. Diepstraten, D. Weiskopf, and T. Ertl. Interactive Cutaway Illustrations. *Computer Graphics Forum*, 22(3):523–532, September 2003.

[EKJ06] B. Erol, Berkner K., and S. Joshi. Multimedia Thumbnails for Documents: Implementation and Demonstration. In *MULTIMEDIA '06: Proceedings of the 14th annual ACM international conference on Multimedia*, pages 487–488, New York, NY, USA, 2006. ACM.

[Fei88] S. K. Feiner. A Grid-Based Approach to Automating Display Layout. In *Proceedings on Graphics interface '88*, pages 192–197, June 1988.

[FLH+06] G. Fuchs, M. Luboschik, K. Hartmann, K. Ali, H. Schumann, and Th. Strothotte. Adaptive Labeling for Interactive Mobile Information Systems. In *10th Int. Conf. on Information Visualisation*, 2006.

[FLM95] A. Frick, A. Ludwig, and H. Mehldau. A Fast Adaptive Layout Algorithm for Undirected Graphs. In *GD '94: Proceedings of the DIMACS International*

Workshop on Graph Drawing, pages 388–403, London, UK, 1995. Springer-Verlag.

[FMHS93] S. Feiner, B. MacIntyre, M. Haupt, and E. Solomon. Windows on the World: 2D Windows for 3D Augmented Reality. In *ACM Symposium on User Interface Software and Technology*, pages 145–155, 1993.

[FP99] J.-D. Fekete and C. Plaisant. Excentric Labeling: Dynamic Neighborhood Labeling for Data Visualization. In *SIGCHI Conf. on Human Factors in Computing Systems*, pages 512–519, 1999.

[FR91] T. M. J. Fruchterman and E. M. Reingold. Graph Drawing by Force-Directed Placement. *Software- Practice and Experience*, 21(11):1129–1164, 1991.

[FRSF06] G. Fuchs, D. Reichart, H. Schumann, and P. Forbrig. Maintenance Support — Case Study for a Multimodal Mobile User Interface. In *IS&T/SPIE's 16th Ann. Symp. Electronic Imaging: Multimedia on Mobile Devices II*, 2006.

[FvDFH90] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics, Priniciples and Practice.* Addison-Wesley Publishing Company, 2nd edition, 1990.

[FW91] M. Forman and F. Wagner. A Packing Problem with Applications to Lettering of Maps. In *Proceedings of the Seventh Annual Symposium on Computational Geometry*, pages 281–288. ACM Press, USA, July 1991.

[GAHS05a] T. Götzelmann, K. Ali, K. Hartmann, and Th. Strothotte. Adaptive Labeling for Illustrations. In *13th Pacific Conf. on Computer Graphics and Applications*, pages 64–66, 196, 2005.

[GAHS05b] T. Götzelmann, K. Ali, K. Hartmann, and Th. Strothotte. Form Follows Function: Aesthetic Interactive Labels. In *Computational Aesthetics 2005. EG WS on Computational Aesthetics in Graphics, Visualization and Imaging*, pages 193–200, 2005.

[GCW94] P. J. Giblin, R. Cipolla, and M. W. Wright. Skeletonization Using an Extended Euclidean Distance Transform. In *Proceedings of the British Machine Vision Conference*, pages 559–568, 1994.

[GGA+06] T. Götzelmann, M. Götze, K. Ali, K. Hartmann, and Th. Strothotte. Practical Illustration of Texts: Customized Search, View Selection, and Annotation. In *Mensch & Computer 2006: Mensch und Computer im StrukturWandel*, pages 437–439, 2006.

[GGA+07] T. Götzelmann, M. Götze, K. Ali, K. Hartmann, and Th. Strothotte. Annotating Images through Adaptation: An Integrated Text Authoring and Illustration Framework. *Journal of the WSCG*, 15(1–3):115–122, 2007.

[GHS05] T. Götzelmann, K. Hartmann, and Th. Strothotte. Labeling Agents. Preprint 11/2005, Department of Computer Science, Otto-von-Guericke University of Magdeburg, December 2005.

[GHS06] T. Götzelmann, K. Hartmann, and Th. Strothotte. Agents-Based Annotation of Interactive 3D Visualizations. In *6th Int. Symp. on Smart Graphics*, pages 24–35, 2006.

[GKG+05] S. Gupta, G. E. Kaiser, P. Grimm, M. F. Chiang, and J. Starren. Automating Content Extraction of HTML Documents. *World Wide Web*, 8(2):179–224, 2005.

[GKU+98] P. Golland, R. Kikinis, C. Umans, M. Halle, M. E. Shenton, and J. A. Richolt. AnatomyBrowser: A Framework for Integration of Medical Information. In *Proceedings of First International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI'98)*, pages 720–731. Springer-Verlag Heidelberg, 1998.

[GNG96] W. H. Graf, S. Neurohr, and R. Goebel. YPPS – A Constraint-Based Tool for the Pagination of Yellow-Page Directories. In U. Geske and H. Simonis, editors, *Proceedings of the KI-96 Workshop on Declarative Constraint Programming, Dresden*, pages 87–97. GMD-Studien Nr. 297, September 1996.

[Gra18] H. Gray. *Anatomy of the Human Body.* Lea & Febiger, Philadelphia, 20th edition, 1918.

[Gra92] W. H. Graf. Constraint-Based Graphical Layout of Multimodal Presentations. In *Int. WS on Advaned Visual Interfaces (AVI)*, pages 365–385, 1992.

[Göt04] T. Götzelmann. Interaktive Visualisierung interner Beschriftungen in 3D-Oberflächenmodellen. Diplomarbeit, Fachhochschule Fulda, FB Angewandte Informatik und Mathematik, February 2004.

[HAS04] K. Hartmann, K. Ali, and Th. Strothotte. Floating Labels: Applying Dynamic Potential Fields for Label Layout. In *4th Int. Symp. on Smart Graphics*, pages 101–113, 2004.

[HGAS05] K. Hartmann, T. Götzelmann, K. Ali, and Th. Strothotte. Metrics for Functional and Aesthetic Label Layouts. In *5th Int. Symp. on Smart Graphics*, pages 115–126, 2005.

[HH88] W. J. Hansen and C. Haas. Reading and Writing with Computers: A Framework for Explaining Differences in Performance. *Commun. ACM*, 31(9):1080–1089, 1988.

[HK02] D. Harel and Y. Koren. Drawing graphs with non-uniform vertices, 2002.

[HNJ+04] S. J. Harrington, J. F. Naveda, R. P Jones, P. Roetling, and N. Thakkar. Aesthetic Measures for Automated Document Layout. In *ACM Symp. on Document Engineering*, pages 109–111, 2004.

[HNJ+06] S. J. Harrington, J. Fernando Naveda, R. P. Jones, N. Sarr, N. A. Thakkar, and P. G. Roetling. System and Method for Measuring and Quantizing Document Quality, August 2006.

[Hod03] E. R. S. Hodges, editor. *The Guild Handbook of Scientific Illustration*. John Wiley & Sons, New York, 2nd edition, 2003.

[HS04] J. Hyldegaard and P. Seiden. My e-Journal - Exploring the Usefulness of Personalized Access to Scholarly Articles and Services. *Information Research*, 9(3), 2004.

[HVA+05] D. Holman, R. Vertegaal, M. Altosaar, N. Troje, and D. Johns. PaperWindows: Interaction Techniques for Digital Paper. In *CHI '05: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 591–599, New York, NY, USA, 2005. ACM.

[Imh75] E. Imhof. Positioning Names on Maps. *The American Cartographer*, 2(2):128–144, 1975.

[Inn03] Innerbody. `http://www.innerbody.com/` 27.12.2004, 1983–2003.

[Ins00] K. Instone. Information architecture and personalization: An information architecture-based framework for personalization systems. `http://argus-acia.com/white_papers/personalization.html`, 2000.

[Ish03] S. Ishizaki. *Improvisational Design - Continuous Responsive Digital Communication*. MIT Press, Cambridge, MA, 2003.

[ITK98] T. Izumi, A. Takahashi, and Y. Kajitani. Air-Pressure-Model-Based Fast Algorithms for General Floorplan. *Design Automation Conference 1998. Proceedings of the ASP-DAC '98. Asia and South Pacific*, pages 563–570, Feb 1998.

[JKS95] R. Jain, R. Kasturi, and B. G. Schunck. *Machine Vision*. McGraw-Hill, 1995.

[JLS03a] C. Jacobs, W. Li, and D. Salesin. Adaptive Document Layout via Manifold Content. In *Workshop on Web Document Analysis (WDA)*, pages 25–28, Edinburgh, 2003.

[JLS+03b] C. Jacobs, W. Li, E. Schrier, D. Bargeron, and D. Salesin. Adaptive Grid-Based Document Layout. *ACM Transactions on Graphics*, 22(3):838–847, 2003.

[JMPS97] R. Johari, J. Marks, A. Partovi, and S. Shieber. Automatic yellow-pages pagination and layout. *Journal of Heuristics*, 2(4):321–342, 1997.

[KGJV83] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecci. Optimization by Simulated Annealing. *Science*, (220):671–680, 1983.

[Käm03] T. Kämpke. Label Placement for Dynamic Objects. In *Machine Graphics & Vision International Journal*, volume 12, pages 215–234. Polish Academy of Sciences, Warsaw, February 2003.

[Knu86] D. E. Knuth. *TEX: The Program*, volume B of *Computers and Typesetting*. Addison Wesley, New York, 1986.

[KP81] D. E. Knuth and M. F. Plass. Breaking Paragraphs into Lines. *Software — Practice and Experience*, 11(11):1119–1184, November 1981.

[KSP97] M. Koivusaari, J. Sauvola, and M. Pietikäinen. Automated Document Content Characterization for a Multimedia Document Retrieval System. In *Proceedings of SPIE Conference on Multimedia Storage and Archiving Systems II*, volume 3229, pages 148–159, Dallas, TX, 1997.

[KT98] K. G. Kakoulis and I. G. Tollis. A Unified Approach to Labeling Graphical Features. In *Proceedings of the 14th Annual Symposium on Computational Geometry*, pages 347–356, 1998.

[LAS04] W. Li, M. Agrawala, and D. Salesin. Interactive Image-Based Exploded View Diagrams. In *Graphics Interface*, pages 203–212, 2004.

[LES95] P. Lüders, R. Ernst, and S. Stille. An Approach to Automatic Display Layout Using Combinatorial Optimization Algorithms. *Software — Practice & Experience*, 25(11):1183–1202, 1995.

[Li02] W. Li. Adaptive Multi-Representation Documents. Master's thesis, University of Washington, 2002.

[LKJ06] Y. S. Lee, H. S. Koo, and C. S. Jeong. A Straight Line Detection Using Principal Component Analysis. *Pattern Recognition Letters*, 27(14):1744–1754, 2006.

[LSC08] M. Luboschik, H. Schumann, and H. Cords. Particle-Based Labeling: Fast Point-Feature Labeling without Obscuring Other Visual Features. *IEEE Transactions on Visualization and Computer Graphics (TVCG) / Proceedings of IEEE Information Visualization (InfoVis'08)*, 14(6), November-December 2008.

[Mac86] Jock D. Mackinlay. Automating the Design of Graphical Presentations of Relational Information. *ACM Transactions on Graphics*, 5(2):110–141, 1986.

[Mac88] J. D. Mackinlay. Applying a Theory of Graphical Presentation to the Graphic Design of User Interfaces. In *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software (UIST '88)*, pages 179–189, 1988.

[Mac92] A. K. Mackworth. The Logic of Constraint Satisfaction. *Artificial Intelligence*, 58(1–3):3–20, December 1992.

[Mas94] T. Masui. Evolutionary Learning of Graph Layout Constraints from Examples. In *UIST '94: Proceedings of the 7th annual ACM symposium on User interface software and technology*, pages 103–108, New York, NY, USA, 1994. ACM.

[May01] R. E. Mayer. *Multimedia Learning*. Cambridge University Press, Cambridge, UK, 2001.

[MB96] J. Müler-Brockman. *Grid Systems in Graphic Design*. Verlag Arthur Niggli, 1996.

[MEA+03] D. Malerba, F. Esposito, O. Altamura, M. Ceci, and M. Berardi. Correcting the Document Layout: A Machine Learning Approach. *Document Analysis and Recognition, 2003. Proceedings. Seventh International Conference on*, 1:97–102, Aug. 2003.

[MF04] Z. Michalewicz and D. B. Fogel. *How to Solve It: Modern Heuristics*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2004.

[MGD+90] B. A. Myers, D. A. Giuse, R. B. Dannenberg, D. S. Kosbie, E. Pervin, A. Mickish, B. Vander Zanden, and P. Marchal. Garnet: Comprehensive Support for Graphical, Highly Interactive User Interfaces. *Computer*, 23(11):71–85, 1990.

[MHN06] T. Maekawa, T. Hara, and S. Nishio. Image Classification for Mobile Web Browsing. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 43–52, New York, NY, USA, 2006. ACM.

[MM99] I. Mani and M. T. Maybury. *Advances in Automatic Text Summarization*. The MIT Press, 1999.

[MM00] K. J. Moll and M. Moll. *Kurzlehrbuch Anatomie*. Urban & Fischer Verlag München. Jena, 16 edition, 2000.

[MMK93] B. A. Myers, R. G. McDaniel, and D. S. Kosbie. Marquise: Creating Complete User Interfaces by Demonstration. In *CHI '93: Proceedings of the INTERACT '93 and CHI '93 conference on Human factors in computing systems*, pages 293–300, New York, NY, USA, 1993. ACM.

[MMT+92] K. Miyashita, S. Matsuoka, S. Takahashi, A. Yonezawa, and T. Kamada. Declarative Programming of Graphical Interfaces by Visual Examples. In *UIST '92: Proceedings of the 5th annual ACM symposium on User interface software and technology*, pages 107–116, New York, NY, USA, 1992. ACM.

[MPS02] G. Mori, F. Paterno, and C. Santoro. CTTE: Support for Developing and Analyzing Task Models for Interactive System Design. *IEEE Transanctions Software Engineering*, 28(8):797–813, 2002.

[MRC91] J. D. Mackinlay, G. G. Robertson, and S. K. Card. The perspective wall: detail and context smoothly integrated. In *CHI '91: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 173–176, New York, NY, USA, 1991. ACM.

[MS91] J. Marks and S. Shieber. The Computational Complexity of Cartographic Label Placement. Technical Report TR-05-91, Center for Research in Computing Technology, Harvard University, 1991.

[MSL99] R. Mohan, J. R. Smith, and C.-H. Li. Adapting Multimedia Internet Content for Universal Access. *IEEE Transactions on Multimedia*, 1:104–114, 1999.

[MW08] Merriam-Webster. Visual dictionary online. `http://visual.merriam-webster.com/`, March 2008.

[Nah] F. Nahrada. A Visionary Visit to a Globe of Villages. `http://rechtsverlag.at/bookandsurf/html/digital_creativity.html`.

[Nas00] J. C. Nash. The (Dantzig) Simplex Method for Linear Programming. *Computing in Science and Engineering*, 2(1):29–31, 2000.

[NPMK97] D. A. Nation, C. Plaisant, G. Marchionini, and A. Komlodi. Visualizing Web Sites using a Hierarchical Table of Contents Browser: WebToc. In *Proceedings of the Third Conference on Human Factors and the Web*, 1997.

[Nun79] G. Nunberg. The Non-uniqueness of Semantic Solutions: Polysemy. *Linguistics and Philosophy*, (3):143–184, 1979.

[OA97] K. O'Hara and Sellen. A. A Comparison of Reading Paper and On-Line Documents. In *CHI '97: Proceedings of the SIGCHI Conference on Human*

Factors in Computing Systems, pages 335–342, New York, NY, USA, 1997. ACM.

[Peu81] D. J. Peuquet. An Examination of Techniques for Remote Formatting Digital Cartographic Data. Part 1: The Raster to Vector Process. *Cartographica*, 18(1):34–38, 1981.

[PGP03] I. Petzold, G. Gröger, and L. Plümer. Fast Screen Map Labeling — Data Structures and Algorithms. In *21st Int. Cartographic Conf.*, 2003.

[Pir99] H. Pirzadeh. Computational Geometry with the Rotating Calipers. Master's thesis, McGill University, Montreal, Quebec, Canada, November 1999.

[Pla81] M. F. Plass. *Optimal Pagination Techniques for Automatic Typesetting Systems*. PhD thesis, Department of Computer Science, Stanford University, 1981.

[PMM97] F. Paterno, C. Mancini, and S. Meniconi. ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. In *Interact'97*, pages 362–369, 1997.

[PMS+98] C. Plaisant, R. Mushlin, A. Snyder, J. Li, D. Heller, and B. Shneiderman. LifeLines: Using Visualization to Enhance Navigation and Analysis of Patient Records. Technical Report CS-TR-3943, 1998.

[PPS99] I. Pitt, B. Preim, and S. Schlechtweg. An Evaluation of Interaction Techniques for the Exploration of 3D-Illustrations. In *Software-Ergonomie'99. Design von Informationswelten*, pages 275–286, 1999.

[PR98] B. Preim and A. Raab. Annotation topographisch komplizierter 3D-Modelle. In *Simulation und Visualisierung*, pages 128–140, 1998.

[PRS97] B. Preim, A. Raab, and Th. Strothotte. Coherent Zooming of Illustrations with 3D-Graphics and Text. In *Graphics Interface*, pages 105–113, 1997.

[PRT97] A. Piolat, J. Y Roussey, and O. Thunin. Effects of Screen Presentation on Text Reading and Revising. *International Journal of Human-Computer Studies*, 47(4):565–589, 1997.

[PS79] J. Pheby and W. Scholze, editors. *The Oxford-Duden Pictorial English Dictionary*. Bibliographisches Institut, Mannheim, 1979.

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

[RBF+02] F. Ritter, B. Berendt, B. Fischer, R. Richter, and B. Preim. Virtual 3D Jigsaw Puzzles: Studying the Effect of Exploring Spatial Relations with Implicit Guidance. In *Mensch & Computer*, pages 363–372, 2002.

[Rec98] W3C Recommendation. Cascading Style Sheets, level 2 CSS2 Specification. `http://www.w3.org/TR/REC-CSS2/`, 1998.

[Rec06a] W3C Recommendation. Extensible Markup Language (XML) 1.0. `http://www.w3.org/TR/xml/`, 2006.

[Rec06b] W3C Recommendation. Extensible Stylesheet Language (XSL) Version 1.1. `http://www.w3.org/TR/xsl11/`, 2006.

[Rec07] W3C Recommendation. XSL Transformations (XSLT) Version 2.0. `http://www.w3.org/TR/xslt20/`, 2007.

[Reh86] R. F. Rehe. *Typography and Design for Newspapers*. Design Research Intl;, 1st edition, June 1986.

[RKB04] C. Rother, V. Kolmogorov, and A. Blake. "GrabCut": Interactive Foreground Extraction Using Iterated Graph Cuts. *ACM Transactions on Graphics*, 23(3):309–314, 2004.

[RM93] G. G. Robertson and J. D. Mackinlay. The document lens. In *UIST '93: Proceedings of the 6th annual ACM symposium on User interface software and technology*, pages 101–108, New York, NY, USA, 1993. ACM.

[Rog92] A. W. Rogers. *Textbook of Anatomy*. Churchill Livingstone, Edinburgh, 1992.

[RP68] A. Rosenfeld and J. Pfaltz. Distance Functions in Digital Pictures. *Pattern Recognition*, 1(1):33–61, 1968.

[RSHS03] F. Ritter, H. Sonnet, K. Hartmann, and Th. Strothotte. Illustrative Shadows: Integrating 3D and 2D Information Display. In *Int. Conf. on Intelligent User Interfaces*, pages 166–173, 2003.

[SB92] M. Sarkar and M. H. Brown. Graphical fisheye views of graphs. In Penny Bauersfeld, John Bennett, and Gene Lynch, editors, *Human Factors in Computing Systems, CHI'92 Conference Proceedings: Striking A Balance*, pages 83–91. ACM Press, Mai 1992.

[Sch89] M. Schmitt. Some examples of algorithms analysis in computational geometry by means of mathematical morphological techniques. In Boissonat, J.D. and Laumond, J.P, editor, *Proceedings Of the Workshop on Geometry and Robotics*, volume Lecture Notes in Computer Science. Springer-Verlag, 1989.

[SCS04] H. Sonnet, M. S. T. Carpendale, and Th. Strothotte. Integrating Expanding Annotations with a 3D Explosion Probe. In *Int. Working Conf. on Advanced Visual Interfaces*, pages 63–70, 2004.

[Sel03] R. H. R. Sellen, A. J.and Harper. *The Myth of the Paperless Office*. MIT Press, Cambridge, MA, USA, 2003.

[SJL08] D. Salesin, C. Jacobs, and W. Li. User Interface for Adaptive Document Layout via Manifold Content. United States Patent 7434164, October 2008.

[SM87] F. Y. Shih and O. R. Mitchell. Skeletonization and Distance Transformation by Gray-Scale Morphology. In *Proc. of SPIE Symposium on Automated Inspection and High Speed Vision Architectures*, pages 80–86, November 1987.

[SM95] K. Sugiyama and K. Misue. A Simple and Unified Method for Drawing Graphs: Magnetic-Spring Algorithm. In *GD '94: Proceedings of the DIMACS International Workshop on Graph Drawing*, pages 364–375, London, UK, 1995. Springer-Verlag.

[SMFBB93] M. Sannella, J. Maloney, B. Freeman-Benson, and A. Borning. Multi-way versus One-way Constraints in User Interfaces: Experience with the Deltablue Algorithm. *Software—Practice and Experience*, 23:529–566, 1993.

[Sof] OPERA Software. Opera for Mobile. `http://www.opera.com/products/mobile/`.

[SPP97] J. Sobotta, R. Putz, and R. Pabst, editors. *Sobotta: Atlas of Human Anatomy. Volume 2: Thorax, Abdomen, Pelvis, Lower Limb*. Williams & Wilkins, Baltimure, 12. English edition, 1997.

[SPP01] J. Sobotta, R. Putz, and R. Pabst, editors. *Sobotta: Atlas of Human Anatomy*. Lippincott Williams & Wilkins, Baltimure, 13. edition, 2001.

[SS99] S. Schlechtweg and Th. Strothotte. Illustrative Browsing: A New Method of Browsing in Long On-line Texts. In *Int. Conf. on Computer Human Interaction (INTERACT-99)*, pages 466–473, 1999.

[SSS97] P. M. Sain, M. K. Sain, and B. F. Spencer. Models for Hysteresis and Application to Structural Control. In *Proceedings American Control Conference*, volume 1, pages 16–20, June 1997.

[Sut63] I. E. Sutherland. Sketchpad: A Man-Machine Graphical Communication System. In E. Calvin Johnson, editor, *Proceedings of the 1963 Spring Joint Computer Conference*, volume 23 of *AFIPS Conference Proceedings*, pages 329–346, Baltimore, MD, 1963. American Federation of Information Processing Societies, Spartan Books Inc.

[T+05] E. R. Tufte et al. Ask E.T.: Mapped Pictures: Image Annotation, 2005. `http://www.edwardtufte.com/bboard/`.

[TB64] A. T. Turnbull and R. N. Baird. *The Graphics of Communication*. Holt, Ribehart & Winston, New York, 1964.

[TBR00] L. Tardif, F. Bes, and C. Roisin. Constraints for Multimedia Documents. In *In Proceedings of the Second International Conference and Exhibition on the Practical Application of Constraint Technology and Logic Programming*, 2000.

[TKSI07] L. Terrenghi, D. Kirk, A. Sellen, and S. Izadi. Affordances for Manipulation of Physical versus Digital Media on Interactive Surfaces. In *CHI '07: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1157–1166, New York, NY, USA, 2007. ACM.

[TMR+97] S. Thad, S. Mann, B. Rhodes, J. Levine, J. Healey, D. Kirsch, R. Picard, and A. Pentland. Augmented Reality through Wearable Computing. *Presence*, 6(4):386–398, August 1997.

[VM] VOXEL-MAN. `http://www.uke.uni-hamburg.de/zentren/experimentelle_medizin/informatik/forschung/vm/index.en.html` 12.04.2004.

[Web] Web Template. http://www.web-hosting-top.com/web-hosting/glossary/terms/41/web-template. Accessed on 02.10.2008.

[Wei91] M. Weiser. The Computer for the 21st Century. *Scientific American*, 265(3):94–104, September 1991. http://www.ubiq.com/hypertext/weiser/SciAmDraft3.html.

[Whi02] A. W. White. *The Elements of Graphic Design: Space, Unity, Page Architecture, and Type*. Allworth Press, 2002.

[WL90] S. Wehrend and C. Lewis. A Problem-Oriented Classification of Visualization Techniques. pages 139–143, 469, 1990.

[WM96] X. Wang and I. Miyamoto. Generating Customized Layouts. In *GD '95: Proceedings of the Symposium on Graph Drawing*, pages 504–515, London, UK, 1996. Springer-Verlag.

[WNA08] E. Wästlund, T. Norlander, and T. Archer. The Effect of Page Layout on Mental Workload: A Dual-Task Experiment. *Computers in Human Behavior*, 24(3):1229–1245, 2008.

[WW93] L. Weitzman and K. Wittenburg. Relational Grammars for Interactive Design. In Ephraim P. Glinert and Kai A. Olsen, editors, *Proc. IEEE Symp. Visual Languages*, pages 4–11. IEEE Computer Society, 1993.

[WW94] L. Weitzman and K. Wittenburg. Automatic Presentation of Multimedia Documents using Relational Grammars. In *MULTIMEDIA '94: Proceedings of the second ACM international conference on Multimedia*, pages 443–451, New York, NY, USA, 1994. ACM.

[YNA99] S. You, U. Neumann, and R. Azuma. Hybrid Inertial and Vision Tracking for Augmented Reality Registration. In *Proceedings of the IEEE Virtual Reality*, pages 260–267. IEEE Computer Society, 1999.

[ZM01] M. Zhou and S. Ma. Representing and Retrieving Visual Presentations for Example-Based Graphics Generation proceedings. In *1st International Symposium on Smart Graphics Hawthorne, NY*, pages 87–94, March 21–23 2001.

Appendix A

Implementation

The prototype software was written in Visual C++ and uses the following libraries:

- Standard Template Library (STL) from SGI (`www.sgi.com/tech/stl`),
- QT: Graphical User Interface Toolkit from TrollTech (`www.trolltech.com`),
- Cassowary Constraint Solver (`www.cs.washington.edu/research/constraints/cassowary/`),
- COIN3D: 3D Graphics Library (`www.coin3d.org`),
- OpenNPAR for non-photorealistic rendering (`http://isgwww.cs.uni-magdeburg.de/research/opennpar/`),
- CGAL 2.4: Computational Geometry Algorithms Library for computing convex hulls and voronoi diagrams (`www.cgal.org`), and
- ViewPoint 3D Model Library.

The implementation was carried on a system with the following specification:

- IBM Lenovo,
- Intel Core 2. T7400 @ 2.16 GHz,
- 1 GB of RAM, and
- Windows XP.

Declaration / Selbstständigkeitserklärung

I herewith declare that I have completed this work by myself and only with the help of the stated references.

Ich erkläre, dass ich die eingereichte Dissertation selbstständig und ohne fremde Hilfe verfasst, andere als die von mir angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Werken wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Kamran Ali

March 31, 2009
Rostock, Germany

Résumé

Personal Data

Name	Kamran Ali
Address	E.-Schlesinger-Str.19/1.3.4.4 18059 Rostock Germany
Date of Birth	July 21, 1977
Place of Birth	Layyah, Pakistan
Nationality	Pakistani

Education

10/2007–07/2009	Ph. D. student at the Department of Computer Graphics, University of Rostock
01/2005–09/2007	Ph. D. student at the Department of Simulation and Graphics, Otto–von–Guericke University of Magdeburg
10/2002–12/2004	M. Sc. in Computational Visualistics Otto–von–Guericke University of Magdeburg
1998–2000	M. Sc. Computer Science, Bahauddin Zakariya University, Multan, Pakistan
1996–1998	B. Sc. Computer Science, Bahauddin Zakariya University, Multan, Pakistan
1982–1996	Higher Secondary Certificate, Govt. Degree College, Layyah, Pakistan

Theses

Thesis 1: *The dynamic requirements of digital media demand that documents are formatted continuously to reflect dynamic changes in both the content and individual users' needs.*

Digital media pose a number of challenges to document presentation. In the digital networked world, the content can be frequently updated by authors, information can be viewed at different display devices, individual clients can request information in different forms and set their own viewing preferences. A document with a fixed design would not be able to achieve these requirements. The dynamic nature of digital media demands that the document layout is determined at client workstation. This cannot be accomplished in real-time by hand for each individual user, therefore, an automated process is necessary to solve the layout problem dynamically.

Thesis 2: *Automated layout of digital documents is a challenging problem that involves complex design issues.*

Creating a document layout requires human skill to design. The design process deals with finding an arrangement of objects in the presentation, subject to space restrictions as well as aesthetic and functional criteria such as symmetry, visual balance, and communication goal. These criteria make automated document layout a challenging task. As a result, the current approaches to document layout are either non-automated or limited in scope.

Thesis 3: *An automated document layout process needs to compute design on behalf of human designer rather than instead of.*

Human factor is important because the design of document should take care of communication goal, aesthetics and cognitive principles. Therefore, it is useful to develop an automated design process whose behaviour could be specified by the human designer. A designer's role in this regard is to construct a set of basic layout templates and specify how the templates should be continuously adapted by the system to solve the design

problems. At client side, the viewing device has its own requirements, such as screen size and resolution. To create the final look of a document, an adaptive document layout system would tailor the designer-made templates to the needs of content and client.

Thesis 4: *Adaptive documents will have limited scope if the design approach is only driven by the template rules, and thus require further treatment.*

A template-based document adaptation approach may work only if the templates are used with the content type they were originally designed for. However, the dynamics of digital documents prevent the designers to predetermine the finished look of the templates for variable content that is unknown at design time. The situation is even more troublesome when the designers have to create templates for varying amount of visually rich content. As a result, the scope of the template design could be either too broad or too narrow. In both situations, a straight-forward layout adaption from the template may not be possible. Therefore, digital documents have to cope with partially-designed templates and with varying amount of illustrated material that needs to be accommodated in the layout.

Thesis 5: *Force-directed techniques can be used to create dynamic design solutions by turning content and layout requirements to forces.*

While the state of the art layout techniques are only influenced by the designer-specified constraints in templates, a novel approach was proposed in this work which also considers content amount and relevance during the layout process. The approach integrates constraint-based layout techniques with force-directed methods to lay out illustrated material in a variety of viewing scenarios. Various content and layout requirements are modeled through a set of physical forces to change the position and size of graphical objects in the page. The forces work hand-in-hand with designer-specified constraints to make adequate changes to the layout.

Thesis 6: *Adaptive annotation layout is essential if the illustrations have to meet the dynamic requirements of digital media.*

Presence of visually rich illustrated material within digital documents means that layout problem is not only limited to the overall appearance of the document page, but also includes the illustrations within the page. Presently, the web browsers provide limited functionality in adapting the layout of illustrated material and embed illustrations as static objects. The readability of such illustrations is vulnerable to layout changes if the illustrations contain textual elements—annotations or labels, insets, legends, and figure

captions. The static illustrations in a digital document exhibit a lack of responsiveness to the changes in content, layout and user interaction. To overcome these issues, the digital documents need to be enhanced with interactive illustrations that set the annotation layout dynamically.

Thesis 7: *Optimal annotation layout is infeasible as the problem has been proven NP-hard.*

The computation of annotation layout can be considered as an optimization problem, as the placement of an individual annotation may have a global effect on the quality of layout. Determining an optimal placement of annotations poses two challenges: (i) There are infinite number of candidate positions where the annotations can be arranged on the picture, which makes evaluating the best positions for annotations NP-hard. (ii) Several aspects that are important to designing hand-made illustrations cannot be precisely modeled through optimization techniques.

Thesis 8: *An automated annotation layout should follow different annotation conventions employed by human illustrators.*

Based on the corpus analysis of hand-made illustrations, a variety of external and internal annotation styles can be identified. These styles help the illustrators to achieve better readability and visual balance in the layouts. By using the layout styles extracted from hand-made illustrations, an automated annotation layout system can improve the usability and appearance of layouts. Importantly, the layout styles dramatically reduce the complexity of layout problem that is computationally tractable for interactive purposes.

Thesis 9: *A unified solution is a fundamental requirement to broaden the scope of automated annotation placement in digital media.*

In order to cope with varying amount of free space in digital documents, it is important that the layout mechanism based on annotation styles is adaptive—it provides transitions between internal and external annotations. Ideally, the method should be developed in a way that it can be used to annotate both 2D and 3D visualization seamlessly. To guarantee a smooth interaction, the annotation layout needs to be computed at interactive rates. Moreover, the changes in the annotation layout should be smoothed over the time to generate a frame-coherent presentation.