An Overview of 3D Software Visualization

Alfredo Teyseyre and Marcelo Campo, Member, IEEE,

Abstract—Software visualization studies techniques and methods for graphically representing different aspects of software. Its main goal is to enhance, simplify and clarify the mental representation a software engineer has of a computer system. During many years, visualization in 2D space has been actively studied, but in the last decade, researchers have begun to explore new 3D representations for visualizing software. In this article, we present an overview of current research in the area, describing several major aspects like: visual representations, interaction issues, evaluation methods and development tools. We also perform a survey of some representative tools to support different tasks, i.e., software maintenance and comprehension, requirements validation and algorithm animation for educational purposes, among others. Finally, we conclude identifying future research directions.

Index Terms—3D software visualization, software comprehension, information visualization, 3D graphics, human-computer interaction.

1 INTRODUCTION

Developing software systems is an arduous task, involving a set of related phases that spawn along the software lifecycle. During all these phases, software engineers need different ways to understand complex software elements. In this context, the use of interactive graphic presentation of data can support significant help to facilitate the analysis and comprehension of such complex information. In fact, experiences in software engineering and visualization areas, confirms that a visual representation of a software system can enhance its understandability and reduce its development costs [1]. Nonetheless, there is a demand for program understanding techniques to graphically represent different aspects of software [2].

The essence of software visualization consists of creating an image of software by means of visual objects. These visual objects may represent, for instance, systems or components, or their run-time behavior. As a result, engineers can obtain an initial perception on how software is structured, understand the software logic, and explain and communicate the development. Effective graphical representations may provide a closer match to the mental model of users than textual representations and take advantage of user’s perception capabilities [3–5]. Actually, the human visual system constitutes a massively parallel processor that provides the highest-bandwidth channel into human cognitive centers [4].

Software visualization in 2D space has been extensively studied. Several authors [6–11] put together collections of papers that reflect the evolution and different categories of the area. In addition, a number of taxonomies of software visualization have been proposed [12–16]. However, as a result of hardware advances, many applications nowadays support 3D graphic capabilities. The inclusion of aesthetically appealing elements, such as 3D graphics and animation, increases the design’s appeal, intuitiveness and memorability of a visualization, but also eases perception of the human visual system [17–19].

Although there is a debate on 2D vs. 3D in the information visualization area [20–22], the use of 3D software visualization has the potential to aid in the development process. 3D software visualization may transform the way that knowledge gathering activities take place during software engineering phases [11, 23]. In this context, this article reports on 3D software visualization work, identifying main directions, techniques, problems and evaluation issues in the area. Our goal is to present an overview of the current state of the art, to provide entry points into the literature and to point out challenges that arise when visualizing software in 3D. The remainder of this paper is organized as follows: First, we introduce the Information Visualization and Software Visualization areas, and we address the definition of terms related to these fields. Then, we provide an overview of the current state of research describing several issues such as 3D visual representations, interaction mechanisms, evaluation methods, and development tools. In addition, we survey some representative works in the area. Finally, we describe future research directions.

2 VISUALIZATION

Card et al. [3] define visualization as “the use of computer-supported, interactive, visual representations of data to amplify cognition”, where cognition is the acquisition or use of knowledge (see Figure 1). These graphical representations can convey complex ideas with clarity, precision and efficiency [24]. In fact, the human visual system (i.e., the eye and the visual cortex of the brain) constitutes an effective parallel processor that supports the maximal communication channel into human cognitive centers [4]. Furthermore, the visual system frees cognitive
capacities by shifting part of processing to it [3]. For example, as a result of visualization, scientists have changed their way of thinking, since they now say they can not do scientific research or communication without visualization [25]. Specially, visualization is a powerful tool that may help users to perform distinct types of cognitive processes [26]:

- **Exploratory**: The user does not know what he is looking for (Discovery).
- **Analytical**: The user knows what he is looking for in the data, trying to determine if it is there (Decision-making).
- **Descriptive**: When the phenomenon represented in the data is known, but the user needs to have a clear visual verification of it (Explanation).

Visualization as a research field is categorized into two major sub-fields: **Scientific Visualization** and **Information Visualization** [3, 4]. Scientific visualization typically represents objects or concepts associated with phenomena from physical world with an inherent spatial component (e.g., chemistry, meteorology, or human body). For example, Figure 2a shows an application (NanoManipulator) that provides a virtual-reality interface to a scanned-probe microscope [27]. On the other hand, information visualization typically involves non-spatial data, that is abstract concepts and relationships (e.g., financial data, bibliographic sources or software). For instance, Figure 2b shows a 3D metaphor (RotaryDiagram) that helps to visualize information evolution [28]. In particular, Software Visualization is a specialized area of Information Visualization, which focuses on improving software comprehension by providing a tangible representation of abstract software concepts. Also, alternative views of the visualization field have been recently proposed, and they may inspire research ideas in hybrid visualization areas [29].

### 2.1 3D Visualization

In short, 3D approaches try to create visualizations that are closer to real world metaphors or to improve space usage by adding an extra dimension. The user is able to rotate and move 3D objects and navigate inside a 3D world. Some approaches propose using a 2D layout seen under a 3D perspective with interaction limited to 2D, that is, a 2.5D approach.
As mentioned in the Introduction, there is a controversial debate on 2D vs. 3D in information visualization area. In order to analyze and identify strengths and weaknesses of 3D/2D, we first review a categorization of 3D visualizations that helps in the analysis [20]:

1) **Augmented 2D views**: Typical 2D visualizations where the third dimension is added just for aesthetic purposes. For example, Figure 3a shows a 3D presentation of a traditional 2D bar chart sorting algorithm [30].

2) **Adapted 2D views**: 2D visualizations extended to 3D to encode additional information. To illustrate this, Figure 3b presents a 3D visual representation of a software release history that displays the structure of the system in 2D planes, and uses the third dimension to display historical information [31].

3) **Inherent 3D application domain views**: This category includes computations involving inherent three-dimensional entities. For instance, Figure 3c represents a software system and its relationships using a *Metaball* metaphor, that is, a 3D modeling technique commonly used to represent complex organic shapes and structural relationships in biology and chemistry [32].

In general, the use of the third dimension in Category 3 is out of discussion. Nonetheless, recent research in specific domains shows that 2D and 3D presentations are useful for different task types, and hence combined 2D/3D displays are suggested [33, 34]. On the other hand, the question of the benefits offered by 3D over 2D still remains in the other categories. Several authors [35, 36] state that when 2 dimensions are enough to show information, it is not desirable to add a third dimension. This extra dimension should be only used to visualize a data set semantically richer. However, other authors think that 3D presentations facilitate perception of the human visual system [17, 18]. They believe that the inclusion of aesthetically appealing elements, such as 3D graphics and animation, can greatly increase a design’s appeal, intuitiveness and memorability of a visualization [19]. For example, Irani and Ware compared 2D UML diagrams to *geon* diagrams (3D shaded solids), and they found out that users can identify substructures and relationship types with much lower error rates for *geon* diagrams than for UML diagrams [18]. In addition, the use of 3D presentations provides a greater information density than two-dimensional ones [37]. For example, an experiment [38] suggests that larger graphs can be interpreted if laid out in 3D and displayed with stereo and/or motion depth cues to support spatial perception. Also, this extra dimension helps to have a clear perception of relations between objects by integration of local views with global views [39] and by composition of multiples 2D views in a single 3D view [18, 40]. Lastly, 3D graphics similarity with the real world enables us to represent it in a more natural way. This means that the representation of the objects can be done according to its associated real concept, the interactions can be more powerful (ranging from immersive navigation to different manipulation techniques) and the animations can be even more realistic.

On the other hand, several problems arise, such as intensive computation, more complex implementation than two-dimensional interfaces, and user adaptation and disorientation. The first problem can be addressed using powerful and specialized hardware. Moreover, development complexity is reduced using several tools like 3D toolkits and frameworks [41–46], 3D modeling languages [47, 48], or 3D software visualization frameworks [44, 49–52].

However, one of the main problems of 3D applications is user adaptation. Most users just have experience with classical *WIMP* (windows, icons, menus, pointing) two-dimensional desktop metaphor. Therefore, the interaction with 3D presentations and possibly the use of special devices demand considerable adaptation efforts to these technologies.

Furthermore, it is often difficult for users to understand 3D spaces and perform actions inside them [53, 54]. In particular, as a consequence of a richer set of interactions and more degrees of freedom, users may be disoriented. For example, Plaisant et al. [55] suggest...
that 3D representations only marginally improve the screen space problem while increasing the complexity of interaction. Moreover, Cockburn and McKenzie [56] evaluated the effectiveness of spatial memory in 2D and 3D, and they found out that navigation in 3D spaces can be difficult and even simple tasks can be problematic. As a way to overcome these limitations 3D enhanced interfaces have been proposed. These interfaces might offer simpler navigation, more compelling functionality, safer movements, and less occlusion than 3D reality [22]. For instance, one alternative to reduce disorientation consists of constraining user navigation with lateral or linear movements [37, 57], or using physical laws such as gravity [58]. Other methods proposed automatic camera assistance during the transition phase from one focus object to the other [59, 60]. In addition, several approaches proposed using landmarks to help users to orient in a 3D world [61–63].

Finally, occlusion may distort the user’s perception of the data-space mainly when the information space is dense [64]. Specially, the occlusion is a serious problem because objects may be occluded and hence appear invisible to the user.

To sum up, there is a vast literature on advantages and disadvantages of 3D vs. 2D with somewhat conflicting results. Table 1 summarizes 3D visualization strengths and weaknesses. Nevertheless, 3D visualizations, if used in ways that exploit their strengths while avoiding their weaknesses [65], may have the potential to aid and improve the development process [11, 23]. In this context, this survey essentially reports results about 3D software visualization.

### 3D Strengths and Weaknesses

<table>
<thead>
<tr>
<th>Strengths</th>
<th>Weaknesses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greater information density</td>
<td>Intensive computation</td>
</tr>
<tr>
<td>Integration of local views with global views</td>
<td>More complex implementation</td>
</tr>
<tr>
<td>Composition of multiple 2D views in a single 3D view</td>
<td>User adaptation to 3D metaphors and special devices</td>
</tr>
<tr>
<td>Facilitates perception of the human visual system</td>
<td>More difficult for users to understand 3D spaces and perform actions in it</td>
</tr>
<tr>
<td>Familiarity, realism and real world representations</td>
<td>Occlusion</td>
</tr>
</tbody>
</table>

Table 1

3D strengths and weaknesses

Software visualization in 2D has been extensively studied and many techniques for representing software systems have been proposed [6, 67]. However, there exists a demand for effective program understanding techniques and methods [2]. In particular, although the question of the benefits offered by 3D over 2D still remains to be answered, a growing area of research is investigating the application of 3D graphics to software visualization with optimistic results [11, 21, 68]. Researchers try to find out new 3D visual representations to overcome some of the limitations of 2D and exploit 3D richer expressiveness. For example, 3D software visualization has been studied in different areas like algorithm animation for educational purposes [20, 30, 69–71], debugging [72], 3D programming [73], requirements engineering [74–76], software evolution [31, 77, 78], cyber attacks [79], ontology visualization and semantic web [80], mobile objects [81], and visualization for reverse engineering, software maintenance and comprehension at different levels of abstraction (source code [21, 82, 83], object-oriented systems [40, 52, 84–89] and software architectures [23, 90, 91]), among others.

### 4 Visual Representations for 3D Software Visualization

One of the problems that software visualization must address is to find an effective tangible representation of something that has no inherent form [10, 11, 23]. In fact, it is crucial to determine which information to visualize, but also, to define an effective representation to convey the target information to the user and support software engineering tasks [15]. Indeed, the design of a software visualization must address a number of different issues, e.g., what information should be presented, how this should be done, what level of abstraction to support, and so on. For example, a tester wanting to
find out who was responsible for a bug and when it was injected will look at version history information. This information may be visualized as a graph [31] or using an abstract representation based on poly cylinders [78]. Many representations for visualizing software have been proposed. For instance, some visual representations are based on abstract shapes, such as graphs [83], trees [44, 92] and geometric shapes [18, 21], and others on real world objects, like 3D cities [93–95], solar systems [96, 97], molecules [98], video games [99, 100], metaballs [32], 3D landscapes [52], and social interactions [75], among others.

Therefore, the main challenge is to develop and evaluate effective mappings, from different aspects of the software to graphical representations, in order to provide insight and easier software comprehension [10, 23]. In that sense, Mackinlay proposes two essential criteria to evaluate the mapping of data to a visual representation: expressiveness and effectiveness [101]. Firstly, expressiveness criteria determine whether a visual representation can express the desired information. Secondly, effectiveness criteria determine whether a visual representation exploits the capabilities of the output medium and the human visual system. Although these criteria were discussed in a 2D graphics context, they can be extended to 3D software visualization. Other researchers discuss desirable properties of visual representations for effective 3D software visualization [23]. These properties may be useful to create new visualizations or to evaluate existing ones.

4.1 Abstract Visual Representations

In this section, we describe several 3D abstract visual representations based on graphs, trees and geometrical shapes. We also analyze their strengths and weaknesses.

4.1.1 Graphs

Many software visualization techniques are based on the graph representation. In short, a graph is a network of nodes and arcs, where the nodes represent entities, such as procedures, objects, classes, or subsystems, while the arcs represent relationships between entities, such as inheritance or method calls. To illustrate, Figure 4 shows a graph view of a software system that we developed and integrated in a 3D Desktop (Project Looking Glass by Sun Microsystems). In particular, for a review on the state of the art in graph visualization in general see [102], graph visualization in information visualization see [103] and combined approaches see [104].

Graph visualization in 2D space, representing software components around simple boxes and lines, has been applied. However, visualization may result completely incomprehensible, not only as software project complexity increases, but also when visualizing multiples attributes of software, even for small projects [11, 68]. This is a consequence of trying to fit large amounts of information into a reduced space. On the other hand, several authors think that larger graph structures can be viewed in 3D [38, 68]. As we mentioned earlier, an empirical study [38] that measured path-tracing ability in 3D graphs suggested that the amount of information that can be displayed in 3D, with stereoscopic and motion depth cues, exceeds by a factor of three a 2D presentation. Moreover, a re-evaluation of the experiment with new display technologies confirmed the previous experiment and showed a much greater benefit than previous studies [105, 106]. However, other authors think that the most successful network visualizations are small ones (e.g., networks with 10-50 nodes and 20-100 links), where users can count the number of nodes and links, and follow each link from source to destination [107–109]. For instance, Shneiderman and Aris proposed a 2D approach using a layout strategy based on user-defined semantic substrates (non-overlapping regions in which node placement is based on attributes) and interaction capabilities to control link visibility [107]. Despite the debate, 3D graphs have been applied in different areas of software visualization, such as software configuration management [77], software architectures [90, 91], and object oriented software [18, 83, 84, 110], among many others. However, several issues must be addressed in order to produce effective 3D visualizations:

- **Layout algorithms**: Graphical layout algorithms try to produce well-organized layouts based on graph properties. These layouts, just as in 2D, are crucial for producing comprehensible 3D visualizations. Although 2D classical layout algorithms can be generalized to 3D, layout of 3D graphs may actually differ. For example, aesthetic constraints, such minimization of edge crossing are less important in 3D because arcs are less likely to intersect [68]. Furthermore, the layout of 3D graphs may be more complex as a consequence of trying to address 3D problems such as occlusions, perspective distortions and so on [104]. In particular, several layouts algorithms do a 2D layout and then extend the graph into 3D using some attributes of the nodes [111] or the third dimension for representing time [40]. For
instance, Figure 5a shows a 2D layout of modules in a message passing system, where the third dimension displays the message sequencing information. Other techniques compose simultaneously several 2D visualization using orthogonal axes [40]. In addition, 3D hyperbolic layouts [112] have been applied. This kind of layouts provides a distorted view of graph, which makes it possible to interact with potentially large graphs, as shown in Figure 5b. Finally, other layout algorithms use the 3D space without preserving a 2D view from some perspective. For example, Force-Directed Methods [102], described in dimension independent terms, can be applied to 3D [113]. These methods model nodes and edges of a graph as physical bodies tied to springs. These bodies have forces acting on or between them, such as magnetic repulsion or gravitational attraction. For instance, Figure 5c shows a 3D UML Class diagram produced using a force-directed layout [114].

- **Node and link representations**: Different colors and shapes may be used to represent several kind of entities, relationships and software metrics. In general, most common representations include spheres for nodes and cylinders for links. Also, more complex representations may help to visualize additional information. To illustrate, Figure 6a presents a class visualized as volume composed of three semi-axes, where each axis, represents a design pattern category [115], that is, behavioral, creational and structural, and each pattern is represented by a distinctive polyhedral shape [116]. Other representations use arbitrarily shaped and colored 3D objects such as Lego bricks (Figure 6b) [90] or 3D primitives called geons (Figure 6c), which take advantage of human ability to remember and distinguish 3D shapes [18].

- **Clustering**: Although 3D enables us to represent graphs with more nodes and links than 2D, when the number of nodes increases noticeable, the scale problem still remains. In order to address this problem, clustering algorithms have been proposed [68]. In short, clustering helps to visualize large graphs by grouping related nodes. For example, the hierarchical composition of software (i.e., subsystem, module, and file) can help to represent large nested graphs of software systems [68, 114].

### 4.1.2 Trees

A tree can represent many software entities, such as subsystems, modules or classes, and relationships between them, such as inheritance or composition. Moreover, since trees have no cycles, unlike graphs, they are generally easy to layout and interpret [3]. In general, tree visualization techniques encode hierarchical information using node-link diagrams, that is explicit relations,
but also containment representations, i.e., relations using space-filling methods. For instance, Figure 7a shows a node-link representation [117]. Figure 7c a containment representation [92] and Figure 7b a mixed approach [85].

A well-known node-link representation is Cone Tree [117,118]. This visualization technique for displaying hierarchical information in 3D can show more information than 2D counterpart. The aim of this technique is to allow a greater amount of information to be navigated and displayed in an intuitive manner. Each sub-tree is laid out as a cone with its root at the top of the cone and the children along the cone base. In addition, nodes are semi-transparent rectangles, and when one has been selected, the tree rotates bringing the nodes on the path to the currently chosen node closest to the user. These nodes are also highlighted. Therefore, the animation and interactive selection enable to create a focal point on the structure and shift some of the cognitive load of comprehending the structure to the human perceptual system. However, one of the main problems of this technique is that some of the nodes are occluded, so it is mainly effective for comprehending the overall structure of the tree.

An alternative to visualize trees, is containment or enclosure. Unlike node link representations, containment fills the space. In particular, 3D enclosure tree representations, such as Information Cube [92], Circle Data Packing [119] and BeamTrees [120], are loosely based upon 2D tree-map visualizations [121]. This kind of representation is effective for showing quantitative variables such as code metrics. For example, Information Cube is a technique to visualize hierarchical information using nested translucent cubes. Due to this transparency, the user is allowed to view the contents of the cubes and their children, while hiding inner information gradually. Meanwhile, Circle Data Packing and BeamTrees, overlap nodes to indicate a parent-child relationship resulting more effective than nested representations for the extraction of global hierarchical information (see Figure 7c).

Finally, other approach called Hierarchical Net 3D [85] mixes containment and explicit relationships to show the static structure of object-oriented systems. The hierarchy of packages, classes, methods and attributes is represented using nested hemispheres for packages, circles for classes, and boxes for methods and attributes. Also, relations between entities, such as dependencies between classes, are represented by explicit connections. Furthermore, the visual complexity is reduced by adjusting the transparency of object surfaces to the distance of the viewpoint.

4.1.3 Abstract Geometrical Shapes

Many software visualization tools use traditional node-link diagrams, but sometimes they present scalability or layout problems. In an effort to explore new representations beyond graphs, several visualization techniques were proposed using abstract three-dimensional geometrical shapes [21,23,89,122].

For example, sv3D [21,82] is a tool that supports the visualization of large-scale software to assist in comprehension and analysis tasks associated with maintenance and re-engineering. It is based on SeeSoft [123] and 3D File Maps [111] techniques. As a result of using the third dimension, texture and abstraction mechanism, this tool can represent higher dimensional data than previous 2D views. To illustrate, Figure 8a shows a container that represents a source file, where each poly cylinder represents a line of text from the source code associated
4.2 Real World

Trying to find suitable representations of software, several researchers proposed using real world metaphors. These techniques use well-understood elements of the world to provide insights about software. For example, some of these techniques are based on a City abstraction [93–95, 125, 126]. In particular, CodeCity [126] represents classes as buildings located in city districts, which in turn, represent packages (see Figure 10a). Another visualization [52] that helps to gain an overview of a software system is presented in Figure 10b. It is based on a 3D landscape technique called ThemeScapes [127]. This visualization shows the relation “is called by” as 3D landscape. Classes are distributed on the plane and their relative distance is determined by their dependencies. The shaded image and height plot encode the frequency of calls. Two hot spots, corresponding to the two peaks in the image, reveal the most called classes in the system (String and ListIter). Moreover, there exist several other systems based on real world metaphors, such as ScenarioML, a requirements engineering tool for validating use cases using social interactions [75], and Metaballs, a 3D modeling technique commonly used to represent complex organic shapes and structural relationships in biology and chemistry [32], among others.

Figure 9. Abstract software representations

with the container. In addition, color is used to represent control structure and height to represent nesting level. On the other hand, a 2D Seesoft representation (Figure 8b) of the same file can just represent nesting level using colors.

FileVis [23] is another visualization tool that shows code files represented individually as floating platforms around a central point, which represents the connectivity of the source code files (see Figure 9a). This system not only shows structural information, but also run-time information using a technique called Callstack to represent the calling structure of C code with colored stacks of blocks [124]. Similarly, another tool shows full stack information of a trace using a 3D spiral representation to encode time series data in a compact and space efficient manner [122]. For instance, Figure 9b shows a sample trace, where color indicates the routine being called, height indicates stack depth, and width indicates total run time.

Figure 10. Real world metaphors
5 Interaction

Software visualization tools not only have to provide effective visual representations, but also effective interactions styles to ease exploration and help software engineers to achieve insight. However, interacting with 3D worlds is more complex than with 2D WIMP interfaces (windows, icons, menus, pointing) [128]. In fact, applications using 3D graphics are essentially different from classical 2D applications and present many challenges. For example, many tasks in 3D applications require the user to manipulate object position and also orientation involving actually six degrees of freedom. Nevertheless, most users have only 2D input devices that should be mapped to a 3D environment. In addition, specialized methods are needed to navigate in a 3D space, due to the increased degree of freedom [60].

In order to perform tasks, interaction techniques provide means for translating user actions into system actions. These interaction techniques can be classified into several categories [54]:

- **Direct Manipulation**: Interaction techniques for manipulating objects provide means to select, position, and rotate objects. Instead of controlling objects through menus or dialogs, users can operate on them. For instance, some systems provide a 3D manipulator or handle [64,82], that is, an interactive 3D object that helps to edit and operate on another object. Other systems provide a virtual hand, a typical approach used in immersive virtual environments, which is in turn intuitive as simulates a real world interaction [54,83].

- **User Navigation**: A film can be used to derive a basic metaphor to build 3D dimensional animated graphics: the 3D geometric objects as the actors, the display as the scenario, the lights and the camera [46]. Specially, the notion of camera helps to observe the scene, focusing the attention on a part of it from a particular position. Also, the camera zoom can be adjusted like the lens of a real camera. Therefore, the user can manipulate (zoom, pan, rotate) the camera interactively to navigate through the scene. To be more precise, navigation is often the primary task in 3D worlds and refers to the activity of moving through them. In this way, navigation provides means to explore and view information from different perspectives and with different degrees of detail. Some systems enable users unconstrained navigation through the information space [83,85,86], however simple 6 degree-of-freedom camera movement through a 3D scene may not be effective in these structures [129]. Others restrict movements in order to lighten possible user disorientation [82,117]. As an illustration, Cone Tree [117] employs a cascading rotation of the 3D cones to bring the desired child nodes to the front. In sv3d [21,82] the visualization can be panned and zoomed in or out, but the position of the camera is fixed. In addition, several systems provide semantic zoom: level of detail changes as user inspects nodes closer. To be specific, as the viewpoint moves towards a particular object, at a certain threshold, the low detailed object representation will be replaced with a high detailed representation [23,91,94]. For example, a system may display a class using a box representation at a high level of detail, and as the user gets closer it may represent additional information such as methods, variables, and software metrics.

- **System control**: The user sends commands to an application. As many of the systems use traditional 2D devices, interaction is done using conventional widgets. Also, other techniques include using 3D widgets [130,131], voice commands [132] and hand gestures [73].

6 Evaluation

Although research on visualization creates impressive images, every design needs to be tested to determine how useful it is for real people doing real tasks [133–135]. In addition, any usability evaluation of a visualization technique has to address both evaluations of visual representations and interactions styles, because exploration may help to achieve insight that a set of fixed images can not [136,137]. In the last years, interest in the evaluation of information visualization systems has grown incorporating human-centered principles of interaction and usability [133–135,138–141].

Most of the methods for usability evaluation were developed for graphical user interfaces and adapted or extended for information visualization. Some well-known methods that have been applied are analytic ones (i.e., expert reviews and cognitive walkthroughs) and empirical evaluations (i.e., controlled experiments, questionnaires, interviews and focus groups). In particular, see [142] for a detail description of these methods. As a result of the lack of evaluation studies in the area of 3D software visualization this section also reports 2D evaluation studies.

Analytic evaluation involves the analysis of the user interface to discover potential usability problems and guide modifications during the development of the system. For instance, Brown et al. reported valuable feedback in the application of expert reviews during the development of a tool to visualize the execution of parallel algorithms and Stasko et al. [143] evaluated an algorithm animation system using cognitive walkthroughs. Furthermore, several researchers performed theoretical analytic studies. For example, Sun and Wong [144] evaluated two common UML tools using fourteen criteria based on gestalt-theoretical principles. Similarly Blackwell et al. [145] analyzed and compared, using a theoretical framework (Cognitive Dimensions Framework), software engineering notations. A number of taxonomies of software visualization have been proposed as well [12–16,146], which in turn provide support for qualitatively evaluation and comparison of software visualizations.
On the other hand, empirical evaluation methods collect usability data by observing or measuring activities of end users interacting with a prototype or an actual implementation of the system. To illustrate, Purchase et al. [147] reported an empirical study of variations in UML notation, Lange and Chaudron [148] evaluated new views to support comprehension of UML models, and Yusuf et al. [149] and Guéhéneuc [150] assessed how people comprehend UML class diagrams using eye-tracking equipment. As mentioned before, Irani and Ware [18, 151] compared 2D UML diagrams to diagrams using 3D primitives called geons and they found out that users can identify substructures and relationship types with much lower error rates for geon diagrams than for UML diagrams. Also, Marcus et al. [152] conducted a usability study to improve a 3D visualization system called sv3D. Despite the overall positive impression that users have of this system, this study reveals their inherent difficulty to adapt to a new technology (3D). Storey et al. [153] evaluated the usability of three user interfaces of a reverse engineering tool by observing users completing a set of software maintenance tasks, followed by a questionnaire and an interview. Also, Jones and Harrold [154] evaluated and compared a technique for fault localization (Tarantula) with other techniques. In addition, Alwis et al. [155] compared three specialized software exploration tools. Finally, several surveys on software visualization based on questionnaires are reported [66, 156].

Indeed, there is a lack of evaluation studies in software visualization research [7] and specially in 3D software visualization. In fact, it is necessary to conduct user studies in order to gain insight, e.g., under what types of tasks and conditions a particular technique is effective. This knowledge is critical because different analytic tasks require different visualization techniques [135]. Furthermore, many of the empirical studies generally include only simple tasks [139]. Additionally, researchers had begun to identify new areas to improve focusing on obstacles that difficult higher-level analytic tasks [157], and new ways to measure and evaluate visualizations based on insight [158].

7 DEVELOPMENT TOOLS

Information and software visualization applications are difficult to build, requiring mathematical and programming skills to implement complex layout algorithms and dynamic graphics [51]. Moreover, the development of 3D graphics applications is a hard work that consumes much more time than conventional 2D graphics applications, requiring specific knowledge about 3D geometry operations as well [41, 46]. Although 3D graphics libraries, such as OpenGL [159], provide a complete application programmer interface (API), they demand significant effort to create even simple visualizations. In particular, most of these APIs require that the application developer cultivate a thorough understanding of a complex programming model, which includes matrix operations, 3D geometry and lighting models [46].

For that reason, several toolkits and frameworks were developed to alleviate these problems. In fact, object-oriented frameworks are a powerful technique to build applications that increases the quality of the software and productivity. In short, a framework represents a reusable design for a specific software domain. To be more precise, using a framework implies the reuse of all the control structure codified in its classes and the production of only the specific code that will be called by the framework. Hence, different applications can be obtained by reusing code and the general design of the application [160]. For example, several frameworks to build 3D graphics applications have been developed: GRAMS [43], GROOP [46], IRIS Inventor [41], and Java3D [42], among others. These frameworks, undoubtedly, make the development of 3D graphics applications easier. Similarly, specific frameworks and toolkits for virtual environments (VE) or distributed virtual environments (DVE) have been also proposed: VR DIVERSE [161], Juggler [162], and a service and component-based framework for DVE [163], among others. In addition, other approaches to develop 3D graphics include 3D modeling languages such as VRML [48] and its successor X3D [47]. These technologies provide a way to model 3D scenes using a description language. In particular, X3D promises to be more successful, due to flexible XML-encoding, modularization through profiles and smarter 3D browsers. Moreover, on the top of X3D and XML, Contigra [45] introduced a component-based approach to construct interactive, three-dimensional applications, either stand-alone or web-based.

Specially, in the area of information visualization, there also exist several toolkits and frameworks. Some of them address just 2D visualizations [51, 164], but others provide 3D graphics capabilities as well [28, 165, 166]. Moreover, software visualization frameworks have been also developed. In general, these frameworks support different metaphors, several layout algorithms, and binding functions. For instance, Mondrian [167] and Evolve [168] display 2D visualizations, Effects renders UML diagrams in three dimensions [169], Luthier [170] helps to visualize object-oriented applications and frameworks, and BLOOM [122] provides support for Box trees, File maps, different graph layouts and Point maps. Additionally, VOUGE [40] concentrates on integrating a number of 2D views to create a more powerful 3D visualization. This framework has been applied in a number of different visualization scenarios: parallel Linda programs, version control and module management, and a C++ class library browser. Other framework, SoftVision [52], supports interactive visualization (2D / 3D) and exploration of the structure, properties, and behavior of component-based systems. Using this framework, developers can freely specify both, the component data to be examined (i.e., what they want to visualize) and how they want to view the data, by assembling pre-packaged components.
such as data editing, filtering, rendering, and user actions. Finally, vizz3D [49, 171] is a framework for creating new software visualizations that enables to configure online views and their mappings instead of hand-coding them.

Researchers also have begun to explore new ways and models to create software visualizations. For example, Bull et al. [172] proposed a model driven approach to assist in the creation of highly customizable interfaces for software visualization. Also, Itoh and Tanaka [173] proposed a component-based framework for generating simple 3D applications that aid users in the use of Web services without the need for programming. The framework uses the 3D media system IntelligentBox as its platform [174]. Moreover, an aspect that is gaining attention, is the integration of visualization tools into development environments (e.g., Eclipse) that developers use every day [169, 175, 176].

8 3D SOFTWARE VISUALIZATION PROJECTS

This section reviews several 3D software visualization systems. It reports on some representative tools for doing different tasks, e.g., algorithm animation, software configuration management, software maintenance and comprehension, and requirements validation, among others. In addition, Table 2 summarizes the main characteristics of the reported tools, based on categories defined by several software taxonomies [14, 15, 146]. These categories include: intent (the purpose of the tool / task supported), audience (who will use the visualization), information (what aspects of the software are to be represented), presentation (characteristics of the output of the visualization), interaction (how the user interacts with visualization), effectiveness (evaluation of the tool). Other relevant information such as associated development tool and year of creation is also reported.

8.1 Algorithm Animation

Algorithm animation visualizes the behavior of an algorithm (data and operations). In fact, animating an algorithm has proven to be useful for education and for research in the design of algorithms [178, 184]. For example, well-known algorithm animation systems such as Polka [20] and Zeus [71] were extended with 3D graphics. Also, JCAT system extends previous work (Zeus) and provides support for web-based algorithm animation in 3D [178]. In particular, GASP, a domain specific animation system, explored 3D animations of computational geometry algorithms [177]. As a consequence of restricting the domain, this system requires much less effort to build animations. To illustrate, Figure 11 shows the animation of a geometric algorithm.

As a result of using the third dimension, it is possible to capture a history of execution (i.e., elementary sorting using classical sticks view extended in z-axis to represent progress), to integrate multiple 2D views to reduce cognitive load (i.e., heapsort 3D view that combines a tree view and a sticks view), to display additional information (i.e., shortest-path animation where the third dimension provides state information about the algorithm as it operates on a data structure represented in 2D), and to visualize inherent 3D domains (i.e., 3D geometric algorithms) [20, 177, 185].

8.2 Software Evolution

Usually a software system changes over time because new services are added, bugs are removed, or the technologies get obsolete. In order to make informed decisions about future developments, it is essential to quickly grasp a comprehensive view of the system evolution [31]. For instance, VRCS [77] helps to do version control and module management using 3D graphics. Each version history is displayed as a 2D tree by taking the z-axis as time. Files (trees) in the same module are placed physically near when looked along z-axis. In addition, module structure and version history are integrated in one 3D visualization. Also, a certain release of the system is represented as a sphere, and versions...
<table>
<thead>
<tr>
<th>System</th>
<th>Audience</th>
<th>Information</th>
<th>Presentation</th>
<th>Interaction</th>
<th>Effectiveness</th>
<th>Development</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Focila [20]</td>
<td>Developers</td>
<td>Algorithms, i.e., sort and graph</td>
<td>2D adapted views</td>
<td>Position and direction of viewpoint</td>
<td>Several examples</td>
<td>Open Toolkit in C++</td>
<td>1993</td>
</tr>
<tr>
<td>Zeus [71]</td>
<td>Developers</td>
<td>Algorithms, i.e., sort and graph</td>
<td>2D adapted views</td>
<td>Zoom, pan, rotate</td>
<td>Several examples</td>
<td>FlexLib</td>
<td>1993</td>
</tr>
<tr>
<td>Gasp [177]</td>
<td>Developers</td>
<td>Geometric algorithms</td>
<td>3D world</td>
<td>VCR metaphor</td>
<td>Several examples</td>
<td>Open Inventor</td>
<td>1995</td>
</tr>
<tr>
<td>JCAT [178]</td>
<td>Developers</td>
<td>Algorithms, i.e., sort and graph</td>
<td>2D adapted views</td>
<td>Zoom, pan, rotate</td>
<td>Several examples</td>
<td>Java3D</td>
<td>2001</td>
</tr>
<tr>
<td>VRCS [77]</td>
<td>Developers</td>
<td>Source code and related attributes</td>
<td>Abstract shapes</td>
<td>Zoom, pan, rotate, 3D handler</td>
<td>Several examples</td>
<td>3D frame-work (OpenIn- ventor)</td>
<td>1997</td>
</tr>
<tr>
<td>EPOsee [180]</td>
<td>Developers</td>
<td>System structure evolution</td>
<td>Partially 3D</td>
<td>Fixed camera</td>
<td>Several examples</td>
<td>Java</td>
<td>2005</td>
</tr>
<tr>
<td>Software Evolution</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software maintenance and comprehension</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code level</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FileVis [23]</td>
<td>Developers</td>
<td>C/Source Code</td>
<td>Abstract Shapes</td>
<td>Zoom, pan, rotate, 3D handler</td>
<td>Several examples</td>
<td>3D frame-work (OpenIn- ventor)</td>
<td>1997</td>
</tr>
<tr>
<td>TraceCrawler [181]</td>
<td>Developers</td>
<td>classes and inheritance object creations and message sends</td>
<td>3D Graph. Structural and behavior integrated view</td>
<td>Zoom, pan, rotate, Virtual hand</td>
<td>Two case studies (700 classes and 6000 events)</td>
<td>3D frame-work (Jun)</td>
<td>2005</td>
</tr>
<tr>
<td>Imsovision [110]</td>
<td>Developers</td>
<td>Classes and inheritance relationships</td>
<td>Abstract shapes</td>
<td>VR environment</td>
<td>Analyzing the seven high level tasks of IV</td>
<td>VRML</td>
<td>2001</td>
</tr>
<tr>
<td>Unified City [125]</td>
<td>Managers</td>
<td>C/C++ / additional software analysis</td>
<td>City metaphor unified single view</td>
<td>Zoom, pan, rotate</td>
<td>Informal use and few examples</td>
<td>3D SV frame-work (Vizz3D)</td>
<td>2007</td>
</tr>
<tr>
<td>Geons3D [18]</td>
<td>Developers</td>
<td>UML Class Diagram</td>
<td>Nodes and links are replaced with 3D primitives (geons)</td>
<td>Work focuses just on graphical representations</td>
<td>Experiments show lower error than UML diagrams.</td>
<td>OpenGL</td>
<td>2003</td>
</tr>
<tr>
<td>X3D-UML [84]</td>
<td>Developers</td>
<td>UML Class Diagram</td>
<td>3D graph</td>
<td>Zoom, pan, rotate</td>
<td>Informal use and few examples</td>
<td>X3D</td>
<td>2005</td>
</tr>
<tr>
<td>Architectural level</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ArchView [90]</td>
<td>Software architect</td>
<td>Software Architecture. Call-tree / part-of from source</td>
<td>Graph and Lego Bricks</td>
<td>Zoom, pan, rotate</td>
<td>Small example application</td>
<td>VRML</td>
<td>1998</td>
</tr>
<tr>
<td>Abacus [91]</td>
<td>Software architect</td>
<td>Sub-system and transaction view, and metrics</td>
<td>Graph Metrics mapped to visual attributes</td>
<td>Zoom, pan , rotate Adaptable Level of detail</td>
<td>A case study (300k LOC)</td>
<td>Not documented</td>
<td>2002</td>
</tr>
<tr>
<td>Requirements Validation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ScenarioML [75]</td>
<td>Stakeholders</td>
<td>Scenarios and use cases</td>
<td>Social interactions of agents in 3D</td>
<td>Fixed Camera</td>
<td>Exploratory study</td>
<td>OpenGL</td>
<td>2006</td>
</tr>
<tr>
<td>ReqViz3D [74]</td>
<td>Stakeholders</td>
<td>Z Specifications</td>
<td>3D World Scripting language</td>
<td>Zoom, pan, rotate</td>
<td>Informal use and few examples</td>
<td>3D frame-work (Java3D)</td>
<td>2005</td>
</tr>
<tr>
<td>PNVIS [182]</td>
<td>Stakeholders</td>
<td>Petri nets</td>
<td>3D World</td>
<td>Zoom, pan, rotate</td>
<td>Informal use and few examples</td>
<td>Java 3D</td>
<td>2004</td>
</tr>
<tr>
<td>Other purposes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Starnine [79]</td>
<td>System Administrators</td>
<td>Cyber attacks</td>
<td>3D multiple metaphor that integrates different views</td>
<td>Fixed Camera</td>
<td>Monitor university/enterprise networks in Japan.</td>
<td>3D Frame-work (Java3D)</td>
<td>2006</td>
</tr>
<tr>
<td>Ontosphere [183]</td>
<td>Developers</td>
<td>Ontology Web services</td>
<td>Graph</td>
<td>Zoom, pan, rotate</td>
<td>Few examples</td>
<td>Java 3D</td>
<td>2006</td>
</tr>
</tbody>
</table>

Table 2: Visualization Tools Summary
composing the release are connected by links (see Figure 12). Similarly, Gall et al. [31] visualized the structure of a system using 3D graphs and represented the historical information by using the third dimension as time. Another tool (cv3D) [78] visualizes information extracted from CVS repositories based on visualization techniques provided by sv3D [21, 82]. At last, other tool that partially supports 3D graphics is EPOsee [180]. This tool provides a 3D bar chart to find dependencies between software components. The third dimension is used to encode additional information of dependencies.

8.3 Software Maintenance and Comprehension

Many 3D visualization tools help software engineers and developers to understand how software works. These tools focus on different level of abstractions (i.e., source code [21, 82], detailed design [40, 52, 84–88] and software architectures [23, 90, 91]) and on different software views (i.e., software structure and/or its run-time behavior). For example, sv3D [21, 82] supports the visualization of large-scale software to assist in comprehension and analysis tasks at code level. This tool represents a line of text from the source code using poly cylinders. Different attributes, such as color, position, height and depth can be associated with software properties and line oriented metrics as well (e.g., color may reflect control structure type and height nesting level).

In addition, many software visualization tools provide an overview of structured or object-oriented systems. For instance, FileVis [23] visualizes systems developed in C using an abstract geometric representation, and Code Mapping [83] uses a 3D graph representation, where nodes representing software artifacts (procedures, variables, calls or data accesses) are mapped to spheres and relationships between software artifacts to lines. In particular, TraceCrawler [86, 181] helps to understand both structure (i.e., classes and inheritance relationships) and behavior of an object-oriented system (i.e., object creations and message sends). This visualization displays the static structure of an application using a graph representation (i.e., classes (nodes) and inheritance relationships (edges)). On the other hand, the application dynamic is represented as object instantiations (boxes) and message sends between objects (connectors). The boxes are displayed as towers of instances on top of their defining classes in the class hierarchy view (see Figure 13). The user can map several metrics to visual attributes of the nodes (i.e., width, length and color). Another system that proposes a unified single visualization, although based on a real world metaphor, is Unified City [125]. This unified single-view visualization supports different kinds of tasks: it allows to see and communicate current development, quality, and costs of a software system quickly (Figure 14). Also, other approach investigates how to improve the interpretation of node-links diagrams using three-dimensional shaded elements instead of simple lines and outlines [18].

Finally, tools such as ArchView [90] and ABACUS [91, 186], focus on architectural abstractions. For example, ABACUS is a tool that creates structural abstractions of software architectures as 3D graph layouts for software maintenance, as seen in Figure 15. This visualization shows architectural structures and relationships present in the code, and uses the color (shading) of the spheres and the size of the cylinders to show metrics-based infor-

---

**Figure 13.** TraceCrawler. Adapted from images courtesy of O. Grevey, M. Lanza, and C. Wysseier. © 2005 IEEE

**Figure 14.** Unified City. Image courtesy of T. Panas, T. Epperly, D. Quinlan, A. Saebjorsen, and R. Vuduc. © 2007 IEEE.
Figure 15. ABACUS: 3D Overall Enterprise Architecture View. The technique is “international patent pending” and has been implemented through the ABACUS product available from Avolution (www.avolution.com.au). Image courtesy of K. Dunsire, T. O’Neill, M. Denford, and J. Leaney. © 2005 IEEE

Figure 17. OntoSphere3D: a multidimensional visualization tool for ontologies. Image courtesy of A. Bosca and D. Bonino. © 2006 IEEE

In conclusion, the third dimension helps to increase information density [82, 86, 125], integrates multiples views (i.e., structure and behavior) [86, 125] and facilitates perception [18].

8.4 Requirements Validation

A correct definition of the requirements of a system has a great impact on the rest of the development process. Thus, to assist software engineers to firm up and explore requirements, several tools were proposed using 3D graphics as a way to validate them as early as possible. For example, PNVIS [76, 182] provides a 3D visualization that helps to analyze the behavior of a system modeled as Petri nets. ScenarioML [75] models a collection of scenarios as social interactions between agents by assigning a character to each actor and entity in these scenarios. During their interaction, actors and agents move to face each other, and speak their actions as a means of expressing their accomplishment. Finally, ReqViz3D [74] is a tool to help users in the process of requirements understanding and validation using 3D visualization techniques. This tool allows specifying the requirements in the formal language Z, defining a graphical representation of them and creating a 3D animated visualization of their execution through which the users can validate them (see Figure 16). To sum up, the application of the third dimension in this area helps to provide familiarity, realism and real world representations [74, 75].

8.5 Other Purposes

Software visualization has also been applied for other purposes such as ontology visualization and semantic web [80, 183], and cyber attacks [79]. For example, Bosca et al. [80, 183] proposed a solution (Ontosphere3D) for the visualization and exploration of ontologies using a 3-dimensional space, as shown in Figure 17. They also reused the visualization component for supporting semantic web applications. Lastly, Starmine [79] is a cyber attack monitoring system that helps system administrators to analyze threats and make a right decision. Using 3D graphics, the system integrates different typical views of cyber threats (see Figure 18): geographical view (to show statistical information of cyber attacks, and geographical location of the source and the destination of each attack), logical view (to see automatic attacks such as scans or Internet worms) and temporal view (to understand transitions of the attacks).

9 CONCLUSIONS

The trend towards more and more visual information is accelerating. There is an explosion of new software visualization techniques that improve software abstractions, create new representations based on real or abstract metaphors, make better layout algorithms, or develop
new navigation mechanisms to browse through complex software representations. Indeed, current advances in visualization are providing new ways to look at software helping software engineers to comprehend, design and analyze software. However, substantial research is still needed to make software tools useful and not just research prototypes. Considering current approaches, discussions and challenges in information and software visualization, and our own experiences as well, we have identified the following areas of research and improvement:

- **Usability**: Software developers want to do their development as quickly and as better as possible. However, many approaches just focus on creating new visualization techniques with little attention on user needs and capabilities. In fact, human factors should play an important role in the design and evaluation of software visualization tools [187].

- **Computer supported collaborative visualization**: Software projects are inherently cooperative requiring many stakeholders to coordinate their activities to produce a software system. In particular, during the last years, software development has also become a geographically distributed activity [188]. Thus, research in collaborative visualization [189] with focus on software may help to improve software engineering process. For example, current technology provided by 3D collaborative virtual worlds for gaming and social interaction may support new ways of working, learning and visualizing software [100, 190].

- **Integration**: Many 3D software visualization systems have been already produced, most of them academic research projects. However, effective SV techniques should be integrated into real working environments, such as Eclipse [176, 180].

- **New display technologies**: As new devices such as high-resolution displays [191] or mobile devices [192] became more common, new techniques must be created or adapted to support device capabilities [134, 193, 194]. For example, take advantage of large screens to simultaneously display more data, but taking into account that some of it will be outside the user’s focal visual field.

- **Widely availability of 3D advanced displays and interactive facilities**: One of the main difficulties of 3D is navigation caused by the discrepancy of using 2D screens and 2D input devices to interact with a 3D world, combined with missing motion and stereo cues [38, 103, 106]. Once we turn into post-WIMP interfaces and adopt specialized hardware (e.g., stereo views, haptic displays, gaze, and motion tracking), 3D techniques may have a substantial effect on software visualization [7].

**ACKNOWLEDGEMENTS**

The authors would like to thank all the people and institutions who have allowed them to use many of the figures present in this paper. The anonymous reviewers deserve special thanks for their effort making so detailed reviews. Finally, we thank the members of the ISISTAN Research Institute (http://www.exa.unicen.edu.ar/isistan/) of the UNICEN University at Tandil, Argentina.

**REFERENCES**


[176] A. Frökk, A. Bruckhoff, and M. Kern, “3D visualisation of code structures in java software systems,” in SoftVis ’06: Proceedings of