SYN: Ultra-Scale Software Evolution Comprehension

Gianlorenzo Occhipinti, Csaba Nagy, Roberto Minelli, Michele Lanza
REVEAL @ Software Institute – USI Università della Svizzera italiana, Lugano, Switzerland

***Abstract*—The comprehension of very large-scale software system evolution remains a challenging problem due to the sheer amount of time-based (*i.e.,* a sequence of changes) data and its intrinsically complex nature (*i.e.,* heterogeneous changes across the entire system source code). It is a necessary step for program comprehension, as systems are not simply created out of thin air in a bang, but are the sum of many changes over long periods of time, by various actors and due to various circumstances.**

We present SYN, a web-based tool that uses versatile visualization and data processing techniques to create scalable depictions of ultra-scale software system evolution. SYN has been successfully applied on several systems versioned on GitHub, including the nearly 20-year history of the Linux operating system, which totals more than one million commits on more than 100k evolving files.

Webpage of the tool and demo video: https://syn.si.usi.ch

***Index Terms*—software evolution, visualization, and analytics**

I. Introduction

Systems constantly adapt to changing requirements [1], making the comprehension of software evolution an inception step for program comprehension activities. The approaches fall into two broad categories: the field of "mining software repositories" (*i.e.,* MSR) and the field of software visualization. MSR deals mainly with approaches to mine non-trivial amounts of information of sometimes complex nature, focusing on ensuring the correctness of the mind. Software visualization uses depictions of the entities under study, which can be two-dimensional (*e.g.,* [2], [3]), three-dimensional (*e.g.,* [4]–[6] and, more recently, even completely immersive (*e.g.,* [7], [8]).

We present an approach implemented in a tool called SYN, which is based on interactive 3D software visualization, and relies, in turn, on a mining infrastructure that we have created to extract the data needed for our purposes.

II. Related Work

First 3D visualization techniques to support software comprehension were proposed in the 1990s [9], [10]. A popular metaphor displays systems as cities [4], [5], [11]–[13]. In 2000, Knight and Munro described Software World, where the entire system is visualized as the world, with cities representing source files with classes in districts, and methods shown as buildings [4]. Panas *et al.* implemented a city metaphor considering both static and dynamic information about programs [11]. Langelier *et al.* used a landscape metaphor to support the quality analysis of large-scale software systems [12]. Wettel and Lanza presented CodeCity to visualize systems in interactive cities with a realistic look, combining layouts, topologies, and metric mappings [5].

Later, they extended CodeCity to visualize the evolution of systems [13]. Steinbrückner and Lewerentz considered software evolution in EVO-STREETS by mapping time to the height of hills on which classes are placed [14]. Their layout, however, is not fully resistent to changes. Scheibel *et al.* proposed a treemap layout algorithm that evolves alongside the changes [2]. Tua *et al.* [3] expanded the work of Scheibel *et al.* using Voronoi treemaps. Pfahler *et al.* took evolution as a first-class concept in M3TriCity by implementing an evolution-resistant layout inspired by the city metaphor [6]. Moreno-Lumbreras *et al.* proposed a web-based implementation of CodeCity that runs both on-screen and in virtual reality (VR) [8], [15]. Making a leap forward, Hoff *et al.* devised a novel immersive environment to explore systems in VR [7].

III. SYN

A. Approach

Software systems can grow notoriously large and accumulate a wealth of historical information throughout their evolution. For example, the Linux kernel[1] had over 1.1M commits, and the core module of LibreOffice[2] had over 480K commits on January 1, 2023.

The challenges of visualizing the evolution of large-scale software systems revolve around dealing with a massive amount of evolutionary data [2], [6], [12], *i.e.,* mining and storing the evolutionary information of the system efficiently and visualizing the collected data to comprehend it.

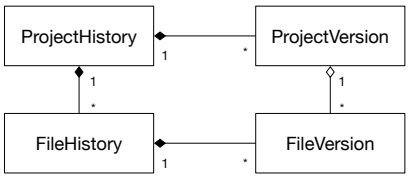

Fig. 1: Evolutionary Model of SYN

1) Mining: SYN implements an evolutionary model inspired by Gîrba [16] (see Figure 1). In the model, *ProjectHistory* represents the history of a repository. It holds *ProjectVersions* and *FileHistories*. *FileHistory* represents a file throughout the evolution of the system, considering its changes, renamings, etc. A change is described by a *FileVersion* that represents a file at a particular point in time (*e.g.,* commit). A *ProjectVersion* is created for each commit and is associated with the related *FileVersion*s.

[1] `https://github.com/torvalds/linux` [acc. 2023/02/27]
[2] `https://github.com/LibreOffice/core` [acc. 2023/02/27]

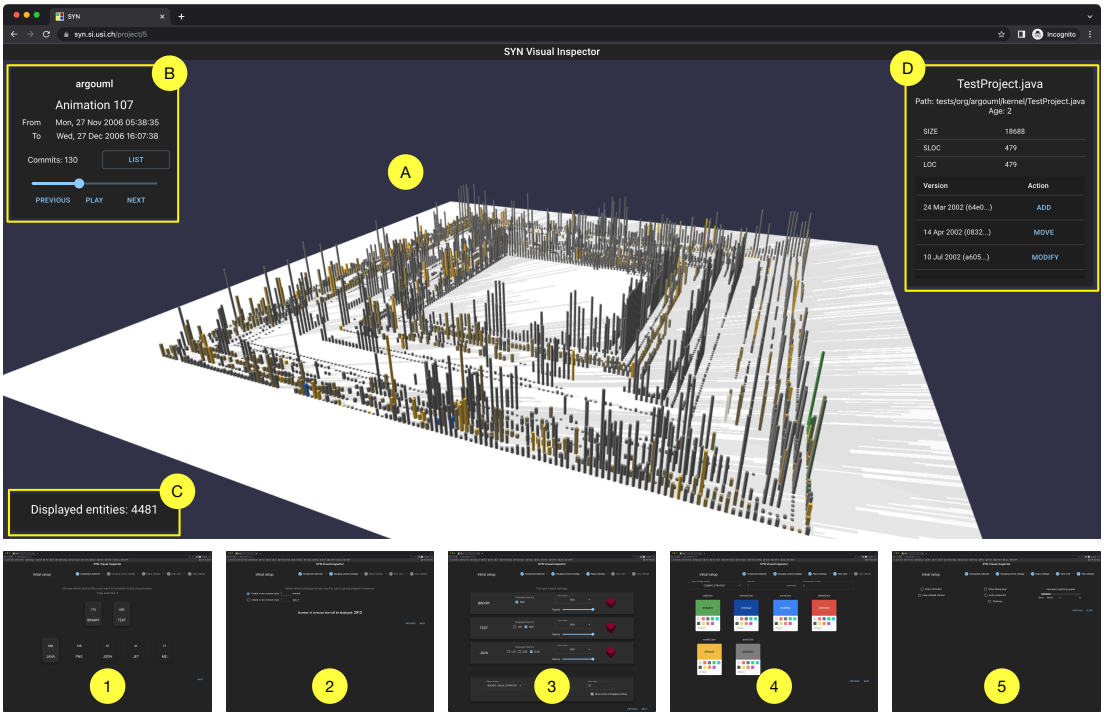

Fig. 2: The Main User Interface of SYN (above) and the Configuration Screens for the Visualization (below)

To build the evolutionary model of a software system, SYN mines the historical information in the project's git repository. It gathers all *file actions*, namely file additions, deletions, content modifications, renames, and moves. Unlike git, SYN considers *rename* as a particular case of a move when the file remains in the same directory.

SYN collects metrics during the analysis for all file modifications. It covers any file in the repository, including source files, textual files, images, and binaries. The metrics include file size (in bytes, for all files), number of lines, number of added/deleted lines (for textual files), and number of non-empty/non-comment source lines (for source files). The metrics are used in the visualization for various mappings.

2) Visualization: SYN uses a three-dimensional representation to depict an evolving software system (see Figure 3). We use the third dimension (*i.e.,* height) to depict metric values, while width and length are constant to ease the layout.

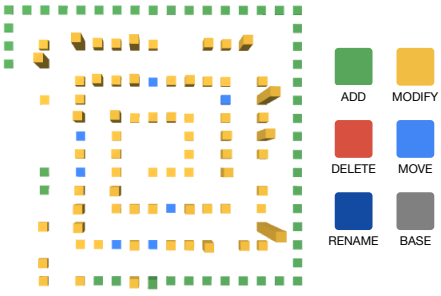

Fig. 3: The Visualization Approach of SYN

SYN uses a simple spiral layout to position the 3D artifacts, similar to the work of Moreno-Lumbreras *et al.* [8], [15].

Each artifact represents a *FileVersion* (*i.e.,* a file at a particular point in time). The color of the artifact depicts the "state" of a *FileVersion* at the visualized point in time, according to the legend in Figure 3. For example, if the file has just been added to the repository, it is green. If the file has been moved within the repository, it turns light blue. If a file is deleted, the artifact is removed from the visualization. To exploit and support the users' spatial memory [17], entities occupy the same position throughout their evolution. Even the space occupied by an entity that has been deleted will not be reused (*i.e.,* it leaves a hole in the visualization). The "Base" color (*i.e.,* grey) denotes that a file has not undergone any change in a user-defined time interval. For example, if we set the time interval to 6 months, all files that have not been changed in more than half a year will be grey. Moreover, as the evolution visualization plays out, any color tends toward the base color (*i.e.,* aging), *e.g.,* if a file is modified once and never again, it will be yellow for one version and slowly fade from yellow to grey as the visualization plays out.

B. User Interface

Figure 2 depicts the web interface of SYN. The visualization (A) takes up the central part of the UI. It is fully interactive, *i.e.,* the user can pan, zoom, rotate, hover and click on the visual entities to obtain additional information in a dedicated panel (D). The panels on the left provide system-wide evolutionary information about the currently displayed version (B) and statistics about the visualization (C). The user uses the top-left panel (B) to traverse the history by either letting the evolution "Play" out automatically as an animation or by stepping into the "Previous" or "Next" snapshot.

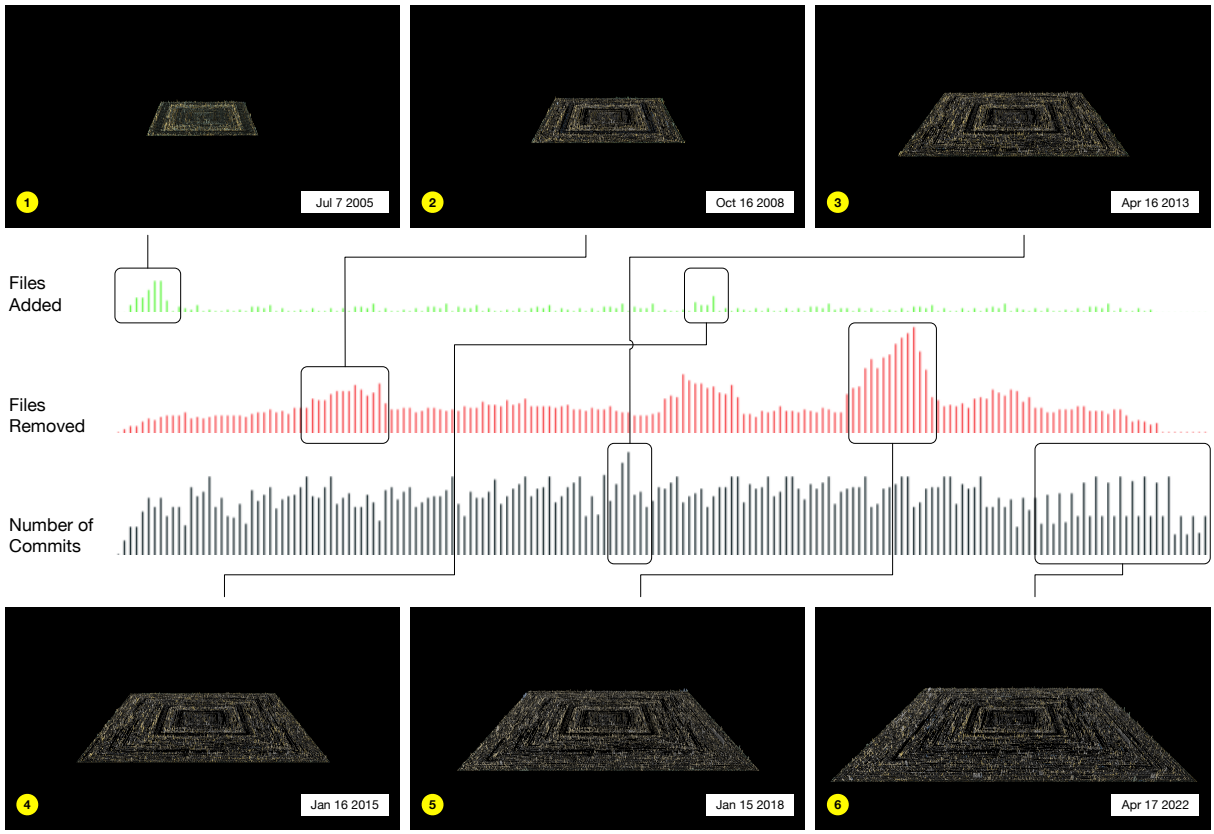

Fig. 4: Linux Evolution Snapshots

Before reaching the visualization, the user is guided through a configuration wizard depicted in the lower part of Figure 2. One can choose (1) the components to visualize (*e.g.,* binary, text, JAVA), (2) the grouping strategy of versions (*e.g.,* by the number of commits or days), (3) the mapping of metrics to shapes and dimensions, (4) the color scheme, and (5) other visualization settings (*e.g.,* VR, shadows).

C. Architecture and Implementation

Figure 5 depicts the overall architecture of SYN, composed of a set of modules as follows.

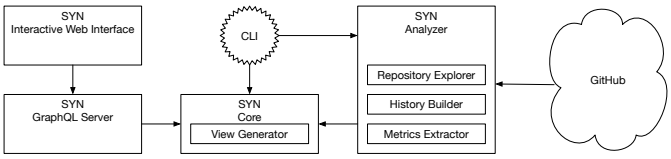

Fig. 5: The Architecture of SYN

The *Interactive Web Interface* implements the main UI as described before. It is a web application written with React.js and uses Babylon.js as a real-time 3D engine. The back end is developed in Java with Spring Boot. Its *GraphQL server* provides endpoints to retrieve the data to be visualized. The *Core* implements the evolutionary model (*ProjectHistory, ProjectVersion, FileVersion, etc.*), and provides means to define and generate new views.

The mining is implemented in the *Analyzer*, which acts as a repository explorer to retrieve data from GitHub, computes the values of specific metrics, and builds the history model of a system. A command line interface (*i.e., CLI*) provides commands to use the *Analyzer* module and simple operations on the mined data (*e.g.,* inspect files in projects).

IV. Case Studies

A. Evolution of Linux

Figure 4 shows six snapshots of the evolution of the Linux GitHub repository. We extracted about 110K *FileHistories*, almost one million commits, and more than two million *FileVersion*s. The first snapshot depicts a short period when many files are being added. This is due to the fact that the history of the Linux GitHub repository is somewhat falsified, as Linus Torvalds created a new repository instead of importing the existing history, as explained in his first commit message.[3] The second snapshot depicts a massive file removal (*i.e.,* the empty band within the spiral). Development activity was always high, as indicated by the bright colors in snapshots 3, 4, and 5. Snapshot 6 shows increased activity, but when looking at the number of commits, we see some regularity regarding how the commits are being performed. Snapshot 6 highlights how massive the system has become.

[3] `https://github.com/torvalds/linux/commit/1da177e4` [acc. 2023/02/27]

B. Evolution of JetUML

Figure 6 shows eight snapshots of the evolution of JetUML,[4] a desktop application for fast UML diagramming. The system has more than seven years of activity and our analysis detected 795 *FileHistories* and 10K *FileVersion*s.

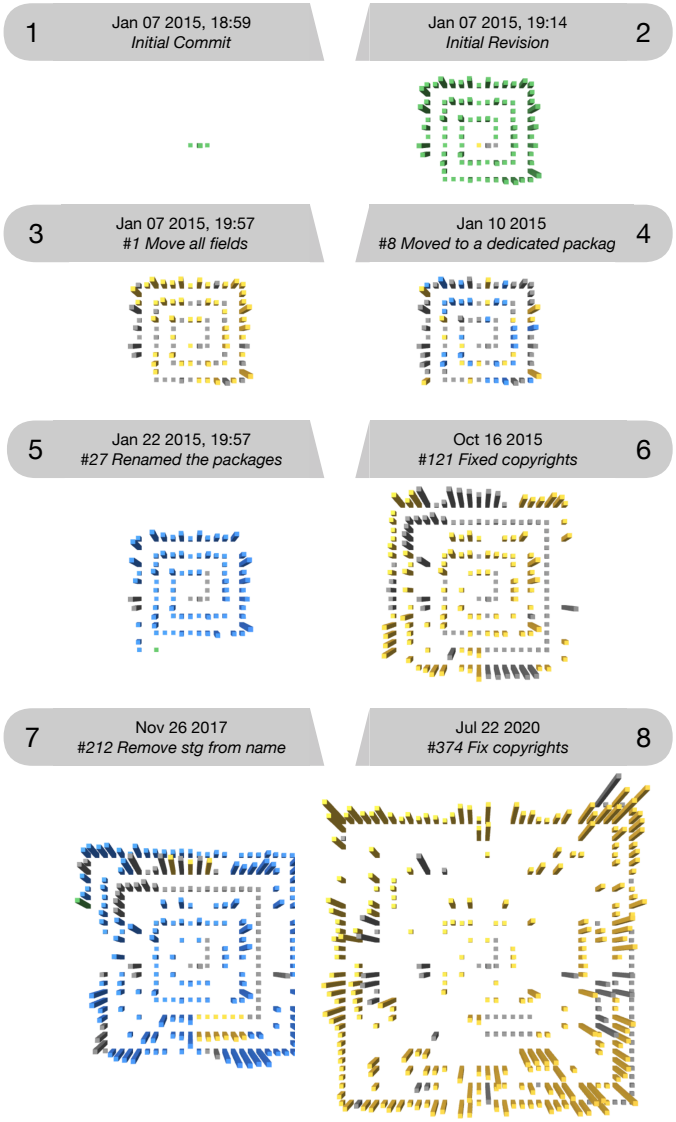

Fig. 6: JetUML Evolution Snapshots

Figure 6.1 depicts the initial commit with three files: README, license, and gitignore. In Figure 6.2 the author committed the initial codebase, composed of 83 files, named *Violet* (*i.e.,* the original name of the system). In Figure 6.3 the author refactored 49 files, changing the position of some fields inside the classes (*i.e.,* modified files are colored in yellow). In the snapshot depicted in Figure 6.4, the author moved some classes from the *Violet* to the *Violetta* folder (*i.e.,* moved classes are represented in light blue). Figure 6.5 marks the birth of the JetUML project, as this is the first time this new name has been used.

[4]**`https://www.jetuml.org`** [acc. 2023/02/27]

Figure 6.6 shows several modified files (*i.e.,* yellow). Our manual analysis showed that these changes are merely the result of a massive copyright update. In Figure 6.8, the author renamed the package `cam.cgill.cs.stg.jetuml` into `ca.mcgill.cs.jetuml`. As a result, we can see 162 light blue entities (*i.e.,* moved entities). Finally, in the snapshot depicted in Figure 6.8 a massive refactoring took place. Once again, this has to do with a change in the copyright of every class. In this snapshot, one can also notice many blank spaces in the middle of the visualization. This means that some earlier entities have been removed and are no longer part of the system.

C. Visualization vs. Animation

It should be noted that SYN, while offering many ways to interact with the visualization, is also used as a tool for software *animation*, where the viewer leans back and observes the system evolve, sees artifacts grow and shrink, and disappear. A viewer can pause the animation at any moment in time, and dive down into the visualization, also being able to access specific revisions on GitHub directly.

V. Conclusion

We presented SYN, a web-based tool to support software evolution comprehension through interactive three-dimensional depictions. SYN is geared towards scalability, and it can process even ultra-large-scale repositories like those hosting Linux. One aspect which is difficult to convey in a written paper is that SYN *maps time on time*, becoming thus a de facto interactive software animation tool.

We illustrated through two example systems how SYN helps the viewer to understand complex evolutionary processes, which can be fine-grained (as in the JetUML example) and small-scale, but also very coarse-grained (as in the Linux example) and ultra-scale.

SYN does not come without limitations: Processing a huge repository can take many hours, and is usually handled offline with the SYN mining infrastructure. Interacting with a visualization within a browser depicting tens of thousands of artifacts sooner than later leads to latency issues outside our control. To overcome this limitation, the present version of SYN visualizes ultra-scale repositories (*e.g.,* Linux) offline using raytracing techniques [18]. While we are confident that hardware advances will keep pushing the boundaries of SYN's online usability, these limits are there to stay for a while.

Future work will be dedicated towards Virtual Reality and Collaborative Visualization & Comprehension.

Acknowledgements

We gratefully acknowledge the support of the Swiss National Science Foundation (SNSF) and the Fonds de la Recherche Scientifique (F.R.S.-FNRS) for the joint Lead Agency project "INSTINCT" (SNF Project No. 190113).

REFERENCES

[1] M. M. Lehman, "Laws of software evolution revisited," in *European Workshop on Software Process Technology*. Springer, 1996, pp. 108–124.

[2] W. Scheibel, C. Weyand, and J. Döllner, "Evocells - a treemap layout algorithm for evolving tree data," in *Proceedings of the 13th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - IVAPP*, INSTICC. SciTePress, 2018, pp. 273–280.

[3] D. P. Tua, R. Minelli, and M. Lanza, "Voronoi evolving treemaps," in *Proceedings of the 9th IEEE Working Conference on Software Visualization (VISSOFT 2021)*, 2021, pp. 1–5.

[4] C. Knight and M. Munro, "Virtual but visible software," in *Proceedings of the 17th International Conference on Information Visualization*. IEEE, 2000, pp. 198–205.

[5] R. Wettel and M. Lanza, "Visualizing software systems as cities," in *Proceedings 4th International Workshop on Visualizing Software for Understanding and Analysis*. IEEE Computer Society, 2007, pp. 92–99.

[6] F. Pfahler, R. Minelli, C. Nagy, and M. Lanza, "Visualizing evolving software cities," in *Proceedings of the 8th Working Conference on Software Visualization (VISSOFT 2020)*. IEEE CS Press, 2020, pp. 22–26.

[7] A. Hoff, L. Gerling, and C. Seidl, "Utilizing software architecture recovery to explore large-scale software systems in virtual reality," in *Proceedings of the 10th IEEE Working Conference on Software Visualization (VISSOFT 2022)*, 2022, pp. 119–130.

[8] D. Moreno-Lumbreras, R. Minelli, A. Villaverde, J. M. González-Barahona, and M. Lanza, "CodeCity: On-screen or in virtual reality?" in *Proceedings of the 9th IEEE Working Conference on Software Visualization (VISSOFT 2021)*. IEEE, 2021, pp. 12–22.

[9] S. P. Reiss, "An engine for the 3D visualization of program information," *Journal of Visual Languages & Computing*, vol. 6, no. 3, pp. 299–323, 1995.

[10] P. Young and M. Munro, "Visualizing software in virtual reality," in *Proceedings of the 6th International Workshop on Program Comprehension (IWPC 1998)*. IEEE Computer Society, 1998, pp. 19–26.

[11] T. Panas, R. Berrigan, and J. Grundy, "A 3D metaphor for software production visualization," in *Proceedings of the 7th International Conference on Information Visualization*. IEEE Computer Society, 2003, pp. 314–319.

[12] G. Langelier, H. Sahraoui, and P. Poulin, "Visualization-based analysis of quality for large-scale software systems," in *Proceedings of the 20th International Conference on Automated Software Engineering*. ACM, 2005, pp. 214–223.

[13] R. Wettel, "Visual exploration of large-scale evolving software," in *Proceedings of the 31st International Conference on Software Engineering (ICSE 2009)*. IEEE, 2009, pp. 391–394.

[14] F. Steinbrückner and C. Lewerentz, "Representing development history in software cities," in *Proceedings of the ACM 2010 Symposium on Software Visualization*. ACM, 2010, pp. 193–202.

[15] D. Moreno-Lumbreras, R. Minelli, A. Villaverde, J. M. Gonzalez-Barahona, and M. Lanza, "CodeCity: A comparison of on-screen and virtual reality," *Information and Software Technology*, vol. 153, p. 107064, 2023.

[16] T. Gîrba, "Modeling history to understand software evolution," Ph.D. dissertation, University of Bern, 2005.

[17] A. L. Shelton and T. P. McNamara, "Systems of spatial reference in human memory," *Cognitive psychology*, vol. 43, no. 4, pp. 274–310, 2001.

[18] A. S. Glassner, *An introduction to ray tracing*. Morgan Kaufmann, 1989.