Management and Visualisation of Non-linear History of Polygonal 3D Models

Jozef Doboš

A dissertation submitted in partial fulfilment
of the requirements for the degree of
Doctor of Engineering
of the
University College London.

Department of Computer Science
University College London

February 7, 2015

To my parents Prof. Jozef Doboš and Ing. Eva Dobošová

I, Jozef Doboš, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

_____

Abstract

The research presented in this thesis concerns the problems of maintenance and revision control of large-scale three dimensional (3D) models over the Internet. As the models grow in size and the authoring tools grow in complexity, standard approaches to collaborative asset development become impractical. The prevalent paradigm of sharing files on a filesystem poses serious risks with regards, but not limited to, ensuring consistency and concurrency of multi-user 3D editing. Although modifications might be tracked manually using naming conventions or automatically in a version control system (VCS), understanding the provenance of a large 3D dataset is hard due to revision metadata not being associated with the underlying scene structures. Some tools and protocols enable seamless synchronisation of file and directory changes in remote locations. However, the existing web-based technologies are not yet fully exploiting the modern design patters for access to and management of alternative shared resources online.

Therefore, four distinct but highly interconnected conceptual tools are explored. The first is the organisation of 3D assets within recent document-oriented No Structured Query Language (NoSQL) databases. These "schemaless" databases, unlike their relational counterparts, do not represent data in rigid table structures. Instead, they rely on polymorphic documents composed of key-value pairs that are much better suited to the diverse nature of 3D assets. Hence, a domain-specific non-linear revision control system *3D Repo* is built around a NoSQL database to enable asynchronous editing similar to traditional VCSs. The second concept is that of visual 3D differencing and merging. The accompanying *3D Diff* tool supports interactive conflict resolution at the level of scene graph nodes that are de facto the delta changes stored in the repository. The third is the utilisation of HyperText Transfer Protocol (HTTP) for the purposes of 3D data management. The *XML3DRepo* daemon application exposes the contents of the repository and the version control logic in a Representational State Transfer (REST) style of architecture. At the same time, it manifests the effects of various 3D encoding strategies on the file sizes and download times in modern web browsers. The fourth and final concept is the reverse-engineering of an editing history. Even if the models are being version controlled, the extracted provenance is limited to additions, deletions and modifications. The *3D Timeline* tool, therefore, implies a plausible history of common modelling operations such as duplications, transformations, etc. Given a collection of 3D models, it estimates a part-based correspondence and visualises it in a temporal flow.

The prototype tools developed as part of the research were evaluated in pilot user studies that suggest they are usable by the end users and well suited to their respective tasks. Together, the results constitute a novel framework that demonstrates the feasibility of a domain-specific 3D version control.

Acknowledgements

This research was sponsored by the Engineering and Physical Sciences Research Council (EPSRC)-funded Engineering Doctorate Centre in Virtual Environments, Imaging and Visualisation (VEIV) (EP/G037159/1), Arup Foresight, Research & Innovation (Job Nº 77322-30), the Rabin Ezra Trust and the Breakthrough Information Technology Exchange (BITE) grant.

Firstly, I would like to thank my primary supervisor, Prof. Anthony Steed, for his invaluable patience, guidance and positive encouragements throughout my studies. My special thanks go to my industrial supervisor, Alvise Simondetti from Arup, as well as my academic supervisor, Prof. Niloy Mitra, for their shared input. Furthermore, I would like to thank Charence Wong and Dr. David Birch from Imperial College London for their conceptual feedback as well as 3D artists Jules Bodenstein, Marcin Klicki, Johnathan Good and Ian Brown for their work on modelling sequences. The Extensible Markup Language 3D (XML3D) integration was jointly developed with Kristian Sons and Dmitri Rubinstein from the German Research Centre for Artificial Intelligence (DFKI) for which I am very grateful.

A particular acknowledgement goes to the guys of the 4.17 office without whom it would have never been so much fun. My fellow classmates helped to resolve compiler and linker errors and always happily and without hesitation offered themselves for the "human trials" presented in this thesis.

Finally, and most importantly, a great thanks goes to my family, friends and especially my partner Zdenka Kissová for their love and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

"Alone we can do so little; together we can do so much."

— Helen Keller

Helen Keller, deafblind from just a few months old, was a prolific American author and political activist. Throughout her life, she was educated and accompanied by Anne Sullivan who herself was blind. Their incredible story and achievements demonstrate the power of *collaboration*, that to this date, despite enormous technological advances more than a century later, remains a challenge for many.

Milestones in computer graphics such as the famous virtual design of *Boeing 777* from early nineties would not have been possible without thousands of people working in parallel over a period of more than five consecutive years. In this project alone, the information had to be communicated between over 6,500 manufacturing personnel, 4,500 engineers and 200 suppliers [Gle98]. It is said that at the peak of the design process, there were 2,200 workstations connected to eight mainframe computers running systems such as the Computer-aided Three-dimensional Interactive Application (CATIA) by Dassault Systèmes just to support the virtual assembly and analysis. These workstations were used by multidisciplinary teams of professionals performing among other things stress testing, weighing, tool design and even training. Although the computational technology and authoring packages have progressed significantly since those days, collaboration and data exchange are pressing issues even 20 years on. As demonstrated by this and many similar examples, some of which are detailed in Chapter 3, the maintenance of information and assets in a large engineering project often involves many authors with different skills and toolsets. Being able to access and visualise massive three dimensional (3D) models is not only important during the design and review stages of a project but it is equally important during commercial exploitation especially when interacting with potential clients and end users. These different groups of stakeholders tend to represent diametrically opposite requirements and preferences that have to be taken into account.

At Arup, a multinational leader in engineering consultancy that sponsored the research presented in this thesis, a virtual design compromises iterative refinements mostly throughout 3D modelling and production stages. More often than not, when developing land or transportational systems, these stages are complicated by local government regulations. For instance, in the United Kingdom (UK), as well as in many other countries around the globe, it is often a legal requirement to host a *public consultation*

during which the members of the general public have the right to voice their concerns and file evidential submissions [Dep05]. This is typically achieved via physical exhibitions yet with very large projects such as the High Speed 2 (HS2) rail, such events incur significant costs and risk not reaching the key stakeholders. Besides, the development plans change frequently and need continual review. In general, the design process is best summarised as a series of overlapping steps as follows:

1. Firstly, standard two dimensional (2D) design drawings are created based on the initial requirements as described by the client.

2. These are then transformed into polygonal 3D models in a cycle of feedback integrating iterations by tens of designers and engineers. During architectural development, such detailed models are placed within the virtual context of existing surroundings, e.g. neighbouring buildings, represented as low polygon geometry.

3. At a point of significant progress, a snapshot of the model is taken for series of simulations and feasibility studies examining diverse factors such as carbon footprint, radiance, noise propagation, etc. By the time these simulations are completed, the model is likely to have evolved significantly making the results only informative.

4. Next, the project is presented to the clients for approval before it is fully unveiled to the public.

5. Suggestions from the public have to be collected and integrated into the final 3D design according to the public consultation results.

6. Finally, the design is manufactured according to the given specification. In case of a built environment, contractors create their own 3D representations for scheduling and plan of works.

To be able to communicate such a project with all of its stakeholders, it is common to create interactive visualisations throughout various stages of the design and development. Massive polygonal 3D scenes are manually subdivided into smaller sectors at natural splitting points, e.g. individual assemblies, fire doors or floor levels, in order to speed up the loading and therefore improve the overall usability of the system. When interacting with geographically diverse teams, the assets have to be transferred either physically on storage media or electronically through shared drives over the Internet. This data is then discussed over the phone while the visualisation is navigated remotely by a designated presenter. The debate usually involves not only the engineers but also clients who are often consulted on specific aspects of the design. Such an approach is clearly not practical in the fast evolving environment of current design processes. What is more, individual designs from disciplines such as heating, ventilating, and air conditioning (HVAC), mechanical, electrical, and plumbing services (MEP), steelwork, etc. have to be federated into an overview visualisation model before decisions can be made. This thesis, therefore, investigates novel means of large-scale polygonal 3D data management in order to support collaboration as well as interactive visualisation that can be shared online without reliance on any specific modelling paradigms. Such an approach is akin to a text-based source code management that revolutionised software development decades earlier, see Chapter 2 for a comparison and analogy.

1.1 Research Problem

The need for collaboration and ultimately version control in line-based 2D computer-aided design (CAD) systems, the direct predecessors of the digital engineering packages of today, was identified as early as the initial commercial spread of the CAD software itself [NH82, KL84, BKK85]. This led to the proposal of the first unifying version control framework designed exclusively for CAD with a public host such as a mainframe or a server storing stable revisions and private clients working on volatile updates [CK86]. Since then, the progress in the capabilities of graphics processing units (GPUs) [OHL+08, BHS13] created the desire for even more detailed and more extensive data representations causing problems particularly in the domains of 3D modelling and visualisation [ZP05]. Whilst engineering or general scientific visualisations have long provided massive datasets, in domains ranging from architectural design to games industry and even 3D printing, there is a growing need to maintain and visualise large polygonal 3D scenes that might be edited by multiple users concurrently [CRS+13, RFH+14]. However, with the ever-increasing data sizes, it becomes more and more difficult to share the models especially with those users who might not have access to the latest rendering hardware. This is certainly the case in cultural heritage that has long been advocating the need for a centralised 3D repository [KFH10].

Still, the standard paradigm of collaborative modelling is sharing of files between various instances of applications because no one tool provides all the functionality that might be required by the industry. A common practice is to open 3D assets in a specific tool, modify them and then re-save those assets again only to repeat the same process in a different application later on. Since most of these applications are single-user and desktop-based, such an approach presents problems not limited to maintaining consistency of the models and dealing with concurrent edits in the same part of a 3D scene. Thus, the management of graphics assets is an important facility in many fields. Although some guidelines and descriptions of the best practices exist [JSE05, Aus06], a few standards, if any, are being implemented. In general, the process can be split between storage on a file system and history tracking of file names and the associated metadata. This usually involves strict naming conventions or fixed file system hierarchies, although, to reduce the storage requirements, a version control system (VCS) can be employed.

Alternatively, even a database (DB) might store file histories. To draw an example from the games industry, if Perforce [Win05] is used to store files and Oracle DB [MH10] to track the history, creating game levels involves querying the DB to get a set of assets, copying them locally and processing them before loading onto the target console. Dedicated tools such as Temerity Pipeline or Alienbrain offer digital asset management and related facilities [JSE05]. Specialised high-end CAD packages such as Dassault Systèmes Enovia [Das12b] or Bentley AssetWise [Ben11a] provide similar functionality, too. The Asset Server in Unity3D [Bla13], for instance, is a networked service for asset management that can utilise various VCSs [Uni10]. However, those systems that are commonly used, e.g. Apache Subversion (SVN) [PCSF08] or Perforce, are better suited to management of text than binary files. Line-based differencing tools make these good at integrating source code but not binary data [HVT98].

What is missing is an open distributed 3D modelling and visualisation framework that would be independent of any one editing tool or a specification. This would achieve unification of design and

version control capable of supporting all existing production pipelines regardless of the industry. The research presented in this thesis, therefore, investigates the advances in recent No Structured Query Language (NoSQL) DBs over their relational counterparts in order to define a practical solution to polygonal 3D version control that would be independent of modelling packages. The first challenge is to define a scalable and extensible core framework for management of non-linear history of 3D scenes constructed from polyhedral models in the cloud. The second challenge is the distributed access to and visualisation of such data over the Internet.

1.2 Research Questions

According to Arup Foresight, Research & Innovation [Sim12], engineering and architectural companies deal with design projects that require collaboration of professionals with different sets of skills and backgrounds. Changes in their 3D models have to be reviewed by project managers, clients and key stakeholders who further suggest design modifications. All of them need the ability to load a large number of assets for remote inspection and analysis. Despite version control being successfully deployed to text-based files for many years, it does not efficiently map to all types of 3D assets. Hence, there is the need for a centralised yet flexible system for management of polygonal 3D data that would integrate version control with direct online visualisation of large models that are likely to evolve over time. Such a system would have to support multiple users, distributed access, flexible changes, efficient storage, access control across projects and revisions, interface with existing packages and should not enforce any particular modelling paradigms on its users. Thus, the work presented in this thesis was guided by research questions as postulated below in order to define a novel system suitable for the likes of Arup. Each of the questions was broken down into specific sub-questions that address the overall research goal.

1. *Can asynchronous collaborative polygonal 3D editing be scaled up to useful model sizes via the application of a domain specific version control system?*

   1.1. *Can a database substitute a traditional file system-based version control to enable efficient tracking of the numerous types of 3D assets?*

   1.2. *Is it possible to sustain collaborative editing of polygonal 3D scenes via interactive user-driven conflict resolution without the need for any asset locking?*

   The first question is addressed by the design and experimental evaluation in Chapters 3 and 4. This work targets the central premise of whether the application of a recent NoSQL DB dedicated to the myriad of 3D assets offers substantial benefits over the existing solutions. The usability of a conflict resolution interface is further supported by a formative user study in Chapter 4.

2. *Can such a specific 3D versioning framework deliver real-time visualisations of large scale 3D scenes over the Internet?*

   2.1. *Can client applications on desktop and mobile platforms connect to the versioning framework directly in order to visualise the revisions of such models?*

2.2. *Can client applications query the system indirectly for revisions and even individual assets in formats that are independent of their underlying data representations?*

The second question is addressed by the experimental prototyping in Chapter 3 and the definition of a Representational State Transfer (REST) architecture for access to 3D assets in Chapter 5. This work extends the revision control framework envisioned in Question 1 to support a rich variety of client applications for remote repository visualisations.

3. *Can a plausible editing provenance be extracted from a sequence of polygonal 3D models?*

3.1. *Can a timeline of high-level editing operations be implied even when no revision metadata is available?*

The last question is addressed by a novel reverse engineering algorithm and a prototype implementation in Chapter 6. Unlike traditional VCSs that record changes only as *added*, *deleted* and *modified*, this approach extracts many more high level semantic modelling operations without the reliance on standard metadata. This is complementary to Questions 1 and 2 by attempting to understand provenance in legacy datasets that would not have been recorded in the novel way.

1.3 Contributions

The main contribution of this thesis is in the definition and implementation of a novel practical set of techniques for version control of disparate 3D assets in a new breed of NoSQL DBs and the improvements over existing solutions this technique brings. While the driving force behind this work was the lack of suitable processes and tools that would support truly collaborative 3D modelling and visualisation on the scale required by the industry, identification of the specific problems and their formulation was equally important. The research, therefore, introduces several novel concepts in the domain of 3D data management and reverse engineering. The contributions of this thesis can be summarised as follows:

1.3.1 Theoretical contributions

- Proposal for a scalable and highly extensible system for management and visualisation of non-linear history of 3D assets supporting branching and merging in Chapter 3.
- Introduction of the concepts of visual 3D differencing and merging as a means of interactive conflict resolution without the need for per asset locking in Chapter 4.
- Definition of a novel REST architecture suitable for the management of version controlled 3D assets over the Internet in Chapter 5.
- Novel algorithm for reverse engineering of a part-based provenance from polygonal 3D models in Chapter 6.

1.3.2 Practical contributions

- Definition of a flexible NoSQL DB schema for version control of decomposed 3D assets including prototype implementation of the system in a desktop, web and mobile clients in Chapter 3.

- Implementation of a prototype 3D Diff tool and its experimental user evaluation in Chapter 4.
- Definition of an open application programming interface (API) implemented into a daemon service and a novel web client application in Chapter 5.
- Prototype implementation of a proposed 3D Timeline tool and its evaluation on six data sequences spanning hundreds of large 3D models as well as a comparative user study in Chapter 6.

1.4 Scope of Thesis

This thesis is concerned with the practical aspects of representation, storage, version control and distributed access to the various types of 3D assets that might have been modelled by multiple users concurrently. In this context, a polygonal 3D model is a hierarchical collection of assets organised in a scene graph data structure as defined in §3.2.1. Although complex 3D representations such as higher-order surfaces like non-uniform rational basis spline (NURBS), boundary representation (BREP), constructive solid geometry (CSG), parametrised models, etc., exist in computer-aided design (CAD), these are not considered within the thesis. Nevertheless, due to the flexibility of the proposed system, in principle, it could be extended to capture these data structures just as well, see Section 7.2 for discussion. Since large polygonal 3D scenes are often comprised of thousands of disparate parts, the focus of the research is on their decomposition and tracking at the high level of scene graph components rather than the low level of vertices. The aim is to define a framework that preserves suitable *delta changes* via non-linear version control in a remote repository regardless of the asset type or the file format. The theoretical contributions, where appropriate, have been demonstrated on prototype implementations as described throughout the thesis. Each technical chapter, therefore, provides system overview and detailed description of the key concepts that guided the design and development of such prototypes.

The 3D repository proposed in Chapter 3 supports all aspects of non-linear version control designed specifically for 3D assets. This system can already track changes on cameras, comments, materials, meshes, textures and transformations. Although animations, bones, higher order surfaces and similar are not currently supported, the system was defined in such a way that it is very easy to extend the implementation with these and any other data representations in the future. Locking functionality was not considered as it is a file system issue. Desktop, web and mobile clients developed in Chapter 3 demonstrate the feasibility of the proposed architecture and the overall principles of version control. In a production environment, it would be appropriate to add plug-in interfaces to popular modelling packages. This, however, is outside the scope of the thesis as the overall goal is to break away from the vendor lock-in of existing modelling software that is so common in the marketplace nowadays. Instead, the framework supports many different 3D file formats that are converted into unified data structures stored in the repository. The visual conflict resolution interface presented in Chapter 4 provides quick merge suggestions by swapping the existing scene graph components in the system. The aim is not to replace editing packages, hence any vertex-level modifications are left for an external editor.

The server-side daemon application defined in Chapter 5 demonstrates the principles of REST style of architecture suitable for version control of 3D assets on the web. During the experimental evaluation,

it was discovered that none of the six tested encoding formats provides the right balance between the number of requests and the speed of decoding. Here, the point was simply to demonstrate the flexibility of the framework to serve representations of 3D assets that are fully independent of the underlying data store rather than defining the most suitable format for delivery of polygonal 3D models over the Internet.

Finally, the system for reverse engineering of editing provenance from consecutive 3D models presented in Chapter 6 estimates a plausible history of possible editing operations. Since different modelling actions might result in the same models, no ground truth comparison was attempted. Instead, the proposed algorithm and the corresponding prototype visualisation tool were evaluated for their speed and consistency on six large and considerably different modelling sequences. The usability of the tool was further evaluated in a comparative user study. Nevertheless, the relative ordering of the detected operations could not be determined as this would require studies into the behaviour of 3D modellers which is not the focus of this thesis.

1.5 Structure of Thesis

Chapter 2 covers related research in the areas of version control, asset management and the analysis of 3D models. This contextualises the work described in this thesis and provides a broad spectrum of existing approaches, some of which motivated the novel solutions proposed in Chapters 3–6. These chapters, therefore, build upon more than 30 years of extensive research in the interrelated fields of polygonal 3D modelling, revision management and data analysis.

Chapter 3 is introductory and presents the main concepts of the management and visualisation of non-linear history of polygonal 3D models using a NoSQL DB. Firstly, the motivation for the research is presented. Functional requirements and system architecture outline the overall design of not only this but all subsequent systems presented throughout the thesis that together form the framework designed specifically for 3D assets. Next, the organisation of data within the system is described and the parallels between a scene graph and a revision history are drawn that enable efficient version control of hierarchical assets in a linear data store. These are supported via revision management operations and their prototype implementation in a desktop, web and mobile clients that are evaluated on diverse polygonal 3D models. The desktop client became the basis for further prototyping in Chapters 4 and 6 while limitations of the direct DB approach were addressed in Chapter 5. Finally, suggestions for extensions are discussed in Section 3.6.

Chapter 4 presents the related concept of 2-way and 3-way visual conflict resolution. The processing pipeline describes how two 3D models and optionally their common ancestor are automatically differenced and then interactively merged into a coherent result. This chapter further introduces the concepts of explicit conflicts resulting from concurrent editing as well as implicit conflicts as the side effects of the merging process itself. All of these concepts were developed into a prototype implementation with support for various visualisation strategies that were evaluated in a formative user study.

Chapter 5 presents a daemon service that overcomes the main limitation of the system presented in Chapter 3. Instead of client applications connecting to the repository directly, this client-server infrastructure provides a gateway system via a REST architecture able to deliver 3D resources in a represen-

tation that is most suitable for the receiving client. System overview outlines the overall design of the service which is built atop of the newly proposed API for version control of 3D assets over the Internet. This is implemented into a prototype application and a corresponding web interface which supersedes the original web client from Chapter 3. The system was evaluated on three increasingly complex 3D scenes using six different encoding formats in some of the most popular web browsers.

Chapter 6 presents a novel algorithm and a visualisation tool for reverse engineering of a part-based provenance from consecutive 3D models. This is especially suitable for legacy dataset that were not created using the version control techniques described in Chapters 3–5. System overview at the beginning of the chapter outlines the processing pipeline stages as follows. Firstly, the system performs an independent segmentation followed by correspondence estimation which is analysed and the results are plotted as an interactive timeline in a prototype application. This was evaluated on six considerably different 3D sequences spanning modelling as diverse as architectural development, CAD prototyping and even free form sculpting. The accompanying user study further evaluates the system for its usability.

Finally, Chapter 7 draws conclusions from this thesis and outlines avenues for future work especially in those areas that were outside the scope of this thesis. The hope is that the findings and suggestions presented here will inspire software vendors in the future in order to embed the proposed techniques into their next versions of 3D authoring tools.

Chapter 2

Background

Fuelled by the ongoing globalisation and the advances in information technology, interactive collaborative environments gained on importance over the past years. This is due to an ever increasing interest in rapid design and development processes that are shared amongst teams scattered all over the world [Joh88, ZS09]. Despite high-end computer-aided design (CAD) and building information modelling (BIM) integrated design tools such as such as Dassault Systèmes CATIA [Das11] or Autodesk Revit [VKR13] providing a form of built-in collaboration via centralised data synchronisations, they are often tied to specific platforms or proprietary file formats. Hence, they do not support the breadth and wealth of the open 3D content generation required by the engineering and creative industries.

Traditionally, such industries rely on highly specialised and diverse authoring tools operated by skilled professionals with many years of experience. However, in these types of environments, collaboration is still facilitated only through the exchange of files across various instances of applications. This poses severe limitations in terms of design and workflow as multiple users are unable to work in parallel. On projects spanning hundreds or even thousands of designers, engineers or contractors, each team uses their own set of tools and processes. These, unfortunately, prevent asynchronous collaboration as large scenes often encompass disparate 3D files that cannot be edited concurrently. Dilemmas that are commonly encountered are how to manage edits that involve multiple objects and how to keep their different revisions synchronised. Thus, the aim of this chapter is to contextualise the research presented in this thesis by reviewing literature as well as existing solutions and best practices that are commonly used.

The background is comprised of four main sections which narrow down the focus of the thesis in a manner most suited to each research question. Section 2.1 explores virtual collaboration through past and current collaborative virtual environments (CVEs) as well as approaches from the construction industry in the UK. Section 2.2 introduces popular solutions for asset management including the main advantages of relying on databases (DBs) over file systems (FSs). Section 2.3 further discusses related concepts of concurrent editing via differencing and merging. Such techniques that span from the fields of software engineering are positioned in the context of 3D graphics including software instrumentation and mesh composition. Finally, Section 2.4 elaborates on asset distribution and different ways of encoding 3D data as well as networked protocols, gaming on demand and 3D map services.

2.1 Virtual Collaboration

In many engineering industries, collaboration and 3D visualisation differ from general purpose computer graphics requirements in several important ways. Firstly, the processing demands and the overall fidelity tend to be much higher especially when representing engineering drawings or architectural 3D models. Simple primitive-based objects or meshes optimised for real-time rendering are neither sufficient nor general enough to support many different industrial concepts. Instead, the engineers have to deal with various 3D editing packages that create problems beyond simple syntactic discrepancies between file formats [Mar07]. Large industrial projects usually depend on a multitude of 3D software that together address various requirements of each task at hand. The tasks include CAD modelling, stress testing, radiance evaluation, carbon emissions, crowd simulations, planning and scheduling of works, etc. [HM04, BKFS13]. However, each demands its own data representations and varying levels of semantic meaning that have to be attached to the geometry. Secondly, specialised engineering software, unlike games, is traditionally single-user without visual depiction of other operators within the same 3D space. Although some authoring tools, e.g. Autodesk Revit [VKR13], enable file synchronisation via their respective remote repositories [Aut10], they are not generally applicable to the next generation of concurrent, design-in-context engineering [Bro14, Aro14].

There are many examples of large-scale computer-supported cooperative works (CSCWs) [BS00]. These include the famous virtual development of the Boeing 777 aeroplane [Gle98] from Chapter 1 or the London King's Cross train station redevelopment [Ove12] described in Chapter 3. Such projects involve multiple companies and contractors with thousands of employees working on the final product delivery. Especially in the architecture, engineering and construction (AEC) industries, the interoperability of the authoring tools is a serious issue [Eas12]. This has been only partially resolved by the Industry Foundation Classes (IFC) [bui13] data format which represents some but not all of the engineering aspects that are required by the construction industry [SFF00, ZYLH14]. The actual concept of generation and management of virtual representations in architectural projects has been proposed a long time ago [EFL+74]. Different software vendors used their own label for the concept such as Virtual Building by Graphisoft [Res03], Integrated Structural Modelling by Bentley Systems [Kuh10] or BIM by Autodesk [Aut03]. Whatever the name, the goal is to enable collaboration and decision making via 3D models. Traditionally, the focus of CAD was on 2D drawings with the support for layers and eventual 3D representations [HB05]. Object-oriented CAD systems replaced 2D lines with 3D elements that could easily capture geometry as well as metadata. Several commercial solutions for collaborative management of engineering models exist. Bentley ProjectWise [Ben11b] is specifically designed for document management in infrastructure projects. This provides file caching servers for geographically distributed teams, dedicated web servers for online access as well as connectors to ArcGIS [ESR99] and Oracle Spatial [Ora12b] systems. This supports search for documents at the file as well as component levels, examination of spatial views for map-based navigation and management of access through complex relationships. In contrast, Autodesk NavisWorks [Aut07] enables federation of multiple engineering models into one visualisation environment for reviewing purposes using functionality such as

Figure 2.1: *Ipswich motorway upgrade project in Arup's Collaborative Map. A web GIS tool is an online portal presenting layered 2D aerial photography, building boundaries and street network in an interactive fashion. Users of the system can collaboratively comment on the proposed development.*

commenting, redlining, measurement takeoffs and clash detection. This further integrates with online solutions such as Autodesk BIM 360 [Aut14b] which further supports multidisciplinary coordination and on-site management. Similar functionality is offered by Autodesk AutoCAD 360 including integration with third party file sharing solutions. However, this stores metadata information in a flat file on a file system (FS). Finally, Graphisoft BIM Server [Gra09] acts as a document store with built-in version control support for the architectural industry. Component-level locking enables multi-user collaboration by sharing updates via a centralised DB. Nevertheless, despite much supporting software already in place, a common approach to version control is to store 3D files on local or shared hard drives with sequential file names based on the current date and incremental revision number. Companies like Balfour Beatty still heavily rely on solutions such as Business Collaborator [UNI11] file sharing system to track various versions of their project documents including 3D models.

In contrast, an example of a transition from file exchange to virtual collaboration is the Ipswich motorway upgrade project by Arup where a custom-made online GIS application Collaborative Map [Ove11] delivered proposed changes to a wide range of stakeholders, see Figure 2.1. Quoting from Arup's Built Environment Modelling (BEM) Handbook [Sim10] (page 37):

> *"WEB GIS: Hosting an interactive GIS model on the Web allows it to be used by a wide range of stakeholders and members of the public, to engage with the design, as well as allowing geographically diverse teams to collaborate on a project."*

Although this tool is able to visualise construction proposals and collect virtual comments by interested parties directly through web browsers, it offers only a top-down 2D overview. Thus, it is not directly applicable to visualisations where virtual polygonal 3D models need to be examined. This tool,

therefore, became the inspiration for the Android app [DSS12] designed specifically for the purposes of public consultation as described in detail in §3.4.5. The need for such a technology is underpinned by the UK Government's Construction Strategy Plan [Gro11] which mandates the use of collaborative 3D technology on all centrally procured construction by 2016. To achieve this, the British Standards Institution specified the requirements for a Common Data Environment (CDE) [The13] to be the single source of information for collection, management and dissemination of all relevant project data. In the long term, stakeholders ranging from commissioners all the way through to the building operators and even demolitioners will need to be able to access, visualise and modify the latest engineering revisions without the need to manage hundreds of disparate 3D files in proprietary tools. In addition, there will have to be an audit trail of changes originating from specific companies and even individuals that can be safely relied upon during legal proceedings. Such requirements are, of course, not limited to the construction industry as other sectors would be able to benefit from a centralised 3D repository, too. The following text, therefore, examines virtual collaboration from the point of view of collaborative virtual environments (CVEs) in §2.1.1 and 3D asset management in Section 2.2.

2.1.1 Collaborative virtual environments

Large-scale distributed 3D environments have long been of interest to the research and design communities [CS98, KVMP12]. Introduced in 1986, a highly influential online role-playing game Habitat by Lucasfilm was the first to coin the term *avatar* to describe a virtual representation of oneself [MF08]. Although not in 3D and very limited by the available technology of the day, the game consisted of a multi-player environment where the host maintained a state of a virtual world and updated all connected clients accordingly. Early examples of distributed 3D worlds include the Simulation Network (SIMNET) [TBT+87, MT95], Distributed Interactive Virtual Environment (DIVE) [CH93] and the Model, Architecture and System for Spatial Interaction in Virtual Environments (MASSIVE) [GB95]. These systems tried to overcome geographical constrains in order to support cooperation between potentially a large number of users. What they had in common was the exchange of information over *peer-to-peer* connections without the need for a centralised control. Each client would effectively broadcast its actions and those interested in this information would process it accordingly. The second important lesson learnt was the understanding that the clients might be different devices with different processing capabilities all trying to interact with the same virtual environment. Other systems such as Minimal Reality (MR) Toolkit [SGLS93] and the WAterloo Virtual Environment System (WAVES) [Kaz93] further exploited parallelism in order to share virtual worlds on low-end platforms via a *client-server* architecture. These systems put even more emphasis on modularity of the design, code reuse and separation of the rendering hardware from the underlying data manipulation in order to maintain interactive framerates. The ability to represent groups of users simultaneously in the same 3D space is said to support the perception of *presence*, *location*, *identity* and *activity* that can be utilised for collaborative purposes [LHM97]. In the *spatial model*, that was implemented into both the DIVE and MASSIVE systems, a virtual space is defined as a volume with boundaries within which various objects reside [BBFG94]. These objects are able to interact with each other at different levels depending on the medium that facilitates the

exchange of information. For example, one might be able to see an object approaching before being able to hear it. Based on these principles, CVEs such as Cybergate [Sch97], Virtual Society [LHM97] and ToolSpace [GS99] enabled rudimentary sharing and manipulation of virtual 3D objects using the Virtual Reality Modeling Language (VRML) [CBM97] directly in web browsers. Their goal was to enable distributed multi-user collaboration in a virtual space that would be scalable and platform-independent.

The term *metaverse* collectively describes the convergence of virtual and physical space as the direct successor to the Internet [Ste92]. In this concept, users personified by avatars interact with the virtual world as well as each other. There are many examples of metaverses [Tho11]. Arguably, the largest of them all, introduced in 2003, is Second Life by Linden Lab [Rym07]. This distributed environment is a multi-user virtual world accessed through a cross-platform desktop-based viewer that heavily depends on the processing and graphical capabilities of the client machines. Its system requirements list different hardware combinations that are necessary to run the viewer application such as NVidia GeForce 6000 series as the bare minimum [Lin10]. On the server side, a virtual region consists of an area of 256×256 meters with a maximum of 50 avatars and 15,000 objects that is governed by a single central processing unit (CPU) core [Tho11]. These regions are organised into a grid-like structure to create vast expanses of land [Au08]. However, the actual 3D objects and textures are stored on independent asset servers, where each asset is referenced by a universally unique identifier (UUID) [Tel08]. The separation of assets from the world itself causes frequent system crashes most likely due to a bottleneck between the servers [Nin08]. The data is live streamed from the cloud to the clients using proprietary communication protocols which prevent proxy caching on the network. This means the same models have to transmitted directly from the asset server rather than an intermediary location. In addition, the built-in 3D editor is primitive-based with a restriction to at most 2^8 components comprising a single complex model. Lately, an experimental support for meshes has been added [Lin11]. These, however, cannot be edited in the world directly. Despite these limitations, the platform was shown to support large-scale collaborations and virtual meetings [Au08], teaching [JTMT09], and even development of whole virtual campuses [DLFPT09]. A similar platform, albeit much less popular, is Active Worlds [HS02].

Open Simulator [Chi09], also known as OpenSim, is an open-source C# derivative of Second Life. Similarly to its predecessor, it is capable of hosting multiple worlds in a stand-alone server simulation or in a grid composed of potentially hyperlinked regions. Founded in 2007, the OpenSim benefited from the Linden Lab's release of its proprietary client under an open source licence. This enabled the community to reverse-engineer the communication system and eventually the server architecture. Although not supporting some of the game-like features of the original, it runs the same protocol as Second Life and can be accessed through the original viewer as well as a number of alternatives. There is already a large number of third-party clients [CCLT+10] that offer varying levels of functionality. OpenSim further supports parametrised and sculpted primitives created either inside the world or, as in the case of sculpted primitives, by a bitmap image within a 3D modelling software. Unfortunately, there is no native support for standard 3D models based on polygon meshes. This problem was further addressed by the realXtend extension which utilises the Ogre3D engine [Jun10] to enable real-time shadows and

more importantly standardised meshes [Ala11]. This project provides its own Qt-based [BS08] viewer as well as an extended server. Scenes can be imported not only in the Ogre file format but also as Collaborative Design Activity (COLLADA) [BF08] objects. However, these implementations have only a limited polygon count support and are, therefore, not capable of rendering large 3D models. In theory, an urban model could be broken into sections and loaded within the grid but this has never been tested.

Another example of an open source CVE is Open Wonderland [KY11], a project originally developed by Sun Microsystems as a showcase for Java 3D capabilities. After Oracle's takeover of Sun, this project continued as a community effort [Kor10]. Similarly to other CVEs, Open Wonderland enables multiple users to interact with each other in a virtual 3D world. Since there is no in-world 3D editor, the project supports 3D meshes via COLLADA imports [Fou10]. In addition, it directly supports 2D X Window System software [SG86, Kil96]. This enables native desktop applications to be placed in 3D environment as texture-mapped polygons. Due to such applications being solely single-user, one has to take control over a window in order to interact with it. In addition, Open Wonderland provides support for shared multi-user 2D and 3D applications development. For example, a built-in slide show viewer allows users to freely browse shared slides independently or synchronised with the presenter. It also supports embedded web browsers that are accessible from within the 3D world.

Finally, a generally more accessible CVE is offered by a cross-platform game Minecraft by Markus Persson [Dun11]. In this pixelated procedurally generated 3D world, everything is made up of textured $1 \times 1 \times 1$ blocks. In the creative mode, users can freely manipulate the world which leads to numerous collaborative creations. This platform was also used for teaching purposes [Sho12] and even as the basis for the Block by Block project with United Nations (UN) attempting to collaboratively develop virtual urban models of the future [Man12]. Minecraft Overviewer [GCBA12] is an open source renderer for Minecraft that generates a slippy map overview interface for large worlds created in the game. In addition, MCEdit [SV12] is an open source world editor that is compatible with the standard Minecraft servers. As of 2014, the Danish Geodata Agency recreated a 1:1 virtual representation of Denmark in Minecraft [Age14]. This took approximately 4,000 billion blocks and over a terabyte of disk space.

In summary, a CVE consists of a collection of 3D objects and tools that populate and manipulate this collection, a communication link which facilitates data transmission, see Section 2.4, and the devices that display the objects. The collection itself can be persistently stored in a DB or a FS, see Section 2.2, or it can be volatile and stored in memory, see §2.4.3. At the core of any such a system is a state-sharing server which updates the connected clients with the latest content of the 3D environment as well as the actions from every user. Alternatively, the clients can exchange information in a peer-to-peer manner, although this is a less popular solution as eventual consistency cannot be guaranteed. What is more, even with a small number of entities, there might simply be too many messages being exchanged at any given time [Kaz93]. The main disadvantage of these systems is their need for a live connection and session broadcasting, although with supporting infrastructure such an approach is proven to scale well.

Even massively multiplayer online games (MMOGs) such as World of Warcraft by Blizzard Entertainment have been shown to encourage virtual collaboration [NH06]. If the world is partitioned into

smaller sections, the client will not require updates from all the other users. Rather, it would need information only about the state of the closest proximity users. Nevertheless, the latency is still an issue which can be tackled by data replication across different geographical locations. This, however, further complicates the design of such a system as messages have to be relayed between the nodes across the network. Still, despite the limited graphics quality and other major limitations of most non-gaming CVEs, the interest in their research continues to this date in areas such as teleconferencing and remote presence [SSO+11, SS12] and even their distribution using modern specifications such as WebGL [BEC+14].

2.2 Asset Management

For relatively small projects, relying on a version control system (VCS) is a popular substitute for purpose-built asset management solutions [SO09]. Especially in demonstrations that include source code or animation scripting, there is a temptation to place everything into a single repository. However, popular VCSs that are being used such as SVN [Cha09] or Git [Cha09] use text-based differencing tools to manage concurrent editing. If the changes take place on different parts of a file, then the intention of both authors can usually be preserved by merging these edits, see Section 2.3 for further details. The Google Docs application suite [Goo12] is another recent example of this type of editing performed in real-time. This system was even adapted for collaborative academic writing [DW06]. In general, however, preserving intentions of edits is more difficult with binary data or more complex scene structures. If a 3D model was stored in a binary format, it is unlikely that two sets of binary changes to the files could be merged successfully. A good example of this shortcoming is the first open source movie Sintel which used SVN for its data and asset management. At the end of the project, the repository reached over 100 GB in size [RVD+11]. What is worse, the team on the project resorted only to linear asset development, see Figure 2.2, due to the lack of suitable 3D differencing and merging functionality that would have been able to identify and resolve conflicting edits. Nonetheless, as discussed in Section 2.3, many important lessons can be learnt from existing tools and techniques.

Whilst proprietary asset management systems such as Bentley AssetWise [Ben11a] and Asset Server in Unity3D [Bla13] are very capable, see Section 1.1, they are either not open, are difficult to implement or deal with 3D data at an inflexible, per-file level. On the other hand, document-based management systems such as Business Collaborator [UNI11] create project extranets in order to offer document and workflow file sharing capabilities. The aim of this thesis is, therefore, to create a domain-specific versioning system with the whole 3D scene and its revisions being stored in a single centralised location. This solution should unify the creation of and access to 3D data as defined in §3.1.1. Such an approach should also save significant storage space over equivalent systems using file-based revision control and provide the ability to manage all asset types and not just the geometry.

2.2.1 File systems

An important debate revolves around the relative merits of using file systems (FSs) versus databases (DBs) for storing large geometric models. As a simplification, FSs provide mechanisms to store, locate and retrieve packaged data, i.e. files, on local or remote devices [TW87]. A distributed file sys-

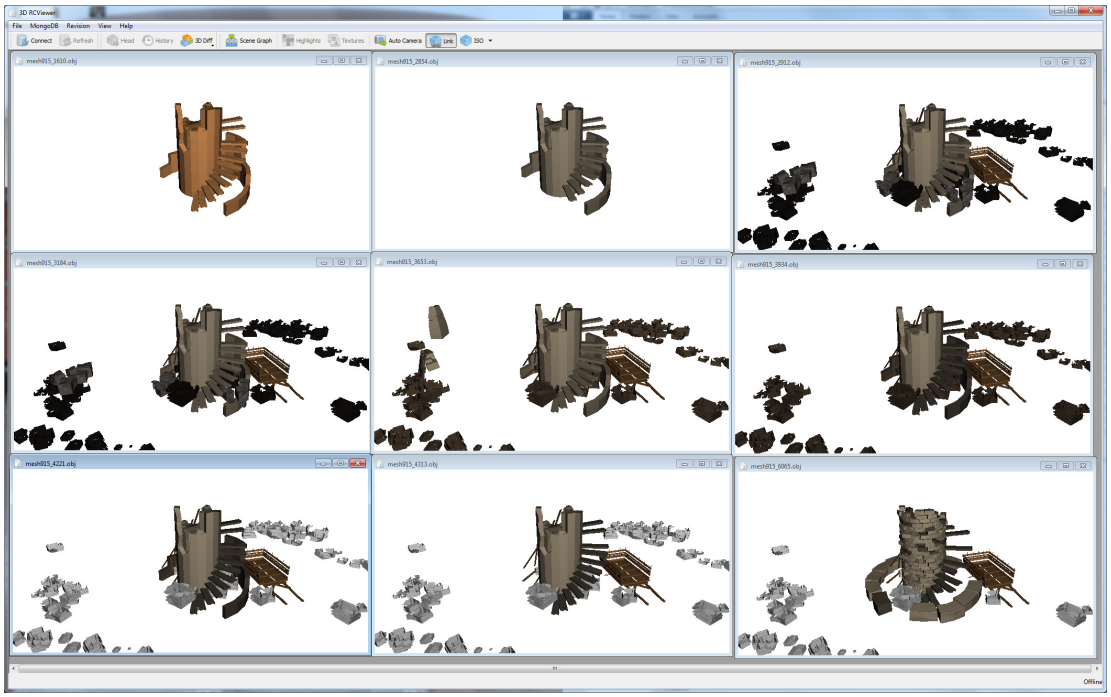


Figure 2.2: *An example of a linear asset development in the open source movie Sintel. In this project, all assets were managed using Apache Subversion (SVN) which was never designed to support 3D data.*

tem (DFS) [Sat93] provides transparency of access to files when the actual storage device is hosted on different servers, whereas a local file system provides fast access to local devices. Whether a file is stored on a local or a distributed system can potentially have a huge impact on the retrieval process and access speeds, although distributed devices are usually used to provide data reliability. Common issues with file systems include how to support user access control and how to enable atomic or concurrent edits. Hence, most file systems come with some form of access control and various mechanisms for managing large storage such as shared names, symbolic links, etc. [Tan07]. Be that as it may, there are several issues with using FSs to support large-scale 3D modelling [LTZH09, KFH10]. The first is version control. Whilst VCSs are very popular, they aggregate edits at a per file level. This means that when assets are changed, the whole files are being modified. However, the scale and type of edits mean that the textual changes might be quite large and pervasive across the file. In the domain of 3D graphics, it is common to have a scene description separate from individual models. There might be a dedicated description file, though some formats such as Extensible 3D (X3D) [JIS+13] can recursively include other files and thus serve both purposes. Another problem with files is that certain modelling operations will require loading of multiple asset files, even though modelling programs tend to only process one asset at a time. If they do support multiple files, this requires careful management, e.g. XRefs in Autodesk 3ds Max [DD13]. This then leads to the another issue which is that the file system itself does not provide the facilities to track assets. Such information must be stored in another form, usually in a format-specific asset management system that serves the roles of both revision control and asset tracking [Aus06]. The third issue is access control which is complicated by the operating system. Unix-like permissions or access control lists can be useful for managing per file data access but these can be difficult to handle remotely [SCFY96].

2.2.2 Databases

There are several potential advantages of using database management systems (DBMSs) [RGG03] over file systems (FSs), especially for the purposes of large-scale 3D modelling. For instance, a database provides centralised control. The database itself might support multiple servers, caching, etc., but logically there is only one point of control [GM08]. In addition, DBs are naturally designed to be used either locally or over the network. Furthermore, the unit of access is under the control of the schema designer. This means that there is flexibility in how 3D data can be stored—it could be represented as a binary blob per mesh or as a single entry per vertex. This choice and its implications are further discussed in Section 3.6. Sophisticated user access control and locks can be integrated into the DB, too. For instance, Oracle Workspace Manager [Ora12c] enables tracking of current, next and historical versions of data on a per row basis via locks on version tables. Databases also support some or all of the atomicity, consistency, isolation and durability (ACID) principles [CB04], so they can preserve data integrity.

CAD databases. In the field of computer-aided design (CAD), very early works introduced the notion of consistent transactions stored in a DB [NH82] and even high-level functional requirements of such a DB [BSF83]. These include multiple views of data using a variety of devices, support for different levels of data representations, easy change management as the work progresses, data distribution over the network, storage and manipulation of geometry and finally support for metadata. Such concepts were later transformed into a detailed functional specification of a CAD DB [SA86]. In order to reduce the disk space requirements when storing versions of large design files, B-tree [Com79] data structures were used to preserve differential files rather than the whole revisions [KL84]. These ideas were later grouped into unifying frameworks for version modelling [CK86, BM88, Kat90]. Still, due to the hardware limitations at the time, the very first architectural walkthrough visualisations were able to support some 8,000 polygons only [Bro87]. These works culminated in standardisation efforts around the Product Data Exchange Standard (PDES) and the STandard for the Exchange of Product Model Data (STEP) formats that were finally merged in 1991 into the International Organization for Standardization (ISO)-STEP [Eas99] specification. For comprehensive accounts of subsequent development in CAD, see [BD94, Ana96]. With the increase in graphics performance and further digitalisation of manufacturing processes, various systems such as relational, object-oriented and even deductive databases based on logic programming were used to store large-scale hierarchical CAD objects [Liu99]. Later works concentrated on searching and data retrieval from these databases including solids [MPSR01, KKM+03], feature based similarities [BKS+05, DA10, BGT+10], shape benchmarks [JKIR06, FGLW08], etc. The research area of CAD databases is vast, see [TV08, DDB11] for general reviews and [BOSD+12, Yag12] for reviews of indexing strategies and formal requirements in CAD/CAM/CAE DBs respectively.

Spatial databases. A special subgroup of DBMSs is that of spatial databases [RSV01]. These DBs originate from the GIS domain [FR13], see Figure 2.1 for an example. They store geographic features such as points, lines, areas, etc. and data about those features such as heights, names and similar [SC03]. Whilst they can store a wide variety of data, they are structured to provide access on a per feature basis. For instance, one might have individual road centre-lines accessible independently in the DB. A few

commercial and open source databases support spatial data directly, in particular Oracle Spatial [Ora12b] and PostGIS [OH11] for PostgreSQL. The Open Geospatial Consortium provides data standards for interchange of spatial data [BMC+96, Ope11]. Although these systems offer flexibility for generic spatial and proximity queries, e.g. [HS99, PTMH05], they also introduce several limitations, the most obvious being a consumption of a significant amount of storage that reduces the performance of the underlying DB [SCR+99, PSZ06]. This has a direct influence on data transmission and introduces the need for large backups and replication. Furthermore, GIS systems are not capable of representing BIM data. 3D files, on the other hand, group every scene or object into a single binary or plain-text representation trading flexibility for storage efficiency. For instance, individual files do not support sub-object queries. Nevertheless, transactional version management in GIS systems using relational database management system (RDBMS) before [AW96]. In application suite Smallworld [ENT11] the transaction control is achieved through recording of changes between states identified by primary keys and the types of change such as insertion, deletion and modification. An alternative is to support map-sheets as an extension to document management [New07]. Hence, when compared to spatial databases, this thesis focuses on distributed access to and editing of general polygonal models suitable for content generation pipelines. Spatial DBs focus mainly on 2D information and metadata, not 3D scenes with properties.

NoSQL databases. In the past few years, the requirement for an efficient data representation and management that would be suitable for the growing needs of the web led to the rise of the NoSQL movement [RW12]. Unlike traditional RDBMS, NoSQL DBs store collections of structured data offering greater flexibility, horizontal scalability and ease of use. In general, NoSQL DBs avoid rigid table structures and tend to be optimised for large read-write operations. Data in these systems is normally stored either as columns, key-value pairs, graphs or documents [HHLD11]. Several recent studies compare performance gains over relational DBs [Cat11, ZYLH14]. Assuming that most polygonal 3D models are commonly represented in a form of scene graphs, i.e. directed acyclic graphs (DAGs), see Section 3.2, a natural fit for their storage might be those DBs that are capable of representing tree and graph-like structures. Such systems store information in nodes with their associated interconnecting edges, see [AG08] for a survey of the field and [DSMBMM+11] for a discussion of their important characteristics. Dedicated graph databases like Pregel [MAB+09] are suitable for large-scale distributed processing that utilises the Map/Reduce paradigm [DG08]. Tasks are performed in parallel so that they can contribute to a single reduction step that generates the results at the end of the processing. Commercial graph management systems such as Neo4j [RWE13], HyperGraphDB [Ior10] and OrientDB [Tes13] represent graphs in file systems (FSs). A study comparing the advantages of Neo4j over relational MySQL [SZT12] when storing DAGs concluded that, although better at string-based searches, this DB would not necessarily be suitable for a production environment due to the way the data has to be structured [VMZ+10]. Another branch of NoSQL advocates the use of document-oriented DBs. Systems such Apache CouchDB [ALS10] and MarkLogic [Zha09] represent data as collections of polymorphic JavaScript Object Notation (JSON)-based [ECM13] documents. MongoDB [MPH10], on the other hand, utilises a Binary JSON (BSON) [Mon14a] specification for efficient data transmission and manipulation

in memory. It also supports full indexing, replica datasets, Map/Reduce and even geospatial indexing that can provide many features similar to spatial databases. Wordnik [Tam10], for example, stores a directed graph of 12 billion documents with an average retrieval of 60 ms in MongoDB.

In summary, graph DBs are suitable for the discovery of relationships between data such as connectivity in social, biological or informational networks. However, a domain-specific solution for the management of non-linear history of polygonal 3D models is unlikely to require graph traversal computations. Instead, it needs an efficient way of encoding various object-like 3D data in a version control repository. Indeed, architectural and engineering models, certainly in the field of building information modelling (BIM), are composed of interconnected objects that capture the shape as well as the properties of a design. Despite all of the aforementioned advantages, there are also arguments against using a database approach altogether. The main one is that loading from a file could be faster than loading from a database. Only a few file formats are described in a way that can be directly mapped to memory; most require some form of single pass or multi-pass parsing to construct a consistent in-memory representation suitable for further processing, see Section 2.4 for examples. This does not mean that the DB access cannot be fast, of course. A DB might store binary data that is intended to be loaded into memory and treated as parts of files for parsing or arrays for passing into the rendering API directly, see Section 5.6.

2.3 Differencing and Merging

The management of versions of documents and other artefacts presents perennial challenges across creative industries, from engineering through to writing. A problem related to asset control and revision management is how users can manage different versions of the same file so that they can then collaboratively edit it [Dou95]. One area where version management has been extremely well studied is software engineering. Most individual programmers or programming teams use some form of VCS to support their development efforts [BKPS97]. Many ways of using VCS in development have been documented, from simply acting as a roll-back mechanism that allows the programmers to revert to an earlier version of the code and discard recent edits across a set of files through to the management of multiple versions of a codebase that might have different functionality such as managing bug fixes on a released version whilst developing a new branch. There is an excellent tool support for VCSs through open source software such as Mercurial [Mac06], SVN [PCSF08] or Git [Cha09], and commercial software such as PerForce [Win05]. All of these systems represent history as a DAG in order to support branching and merging [Bau08]. Modern integrated development environments (IDEs) such as Visual Studio [RGAM10], Eclipse [Bur05], XCode [Wen14], Qt Creator [Ris13], etc. can accommodate VCSs directly. Most VCSs that are supported enable distributed access to the repository so that multiple editors can modify the files and one editor can operate on multiple version sets or computers. This then opens up the risk that different copies of the files will be edited in conflicting ways. Dealing with conflicts is one of the main functions of a VCS. The more general issues of dealing with software changes have stimulated independent journals and conferences, see [BMZ+05] for a taxonomy. Despite attempts to automate propagation of software changes, manual intervention is necessary to verify change integration except in very specific controlled situations such as source code refactoring.

Surveys of software merging [Men02, ASW09] identified several concepts that are useful when considering versioning of 3D models as follows:

1. First is the observation that in *pessimistic* version control, conflicting edits are mostly avoided by locking specific files or assets in the VCS. An analogy in a 3D model would be to lock specific asset files to prevent editing. However, in both the source code and in the 3D cases, pessimistic version control assumes that the changes are local. This might be the case if the meshes, textures, etc. in the 3D model were not reused elsewhere in the same scene. Pessimistic version control is, thus, hard to manage and does not scale well. In *optimistic* version control [BSV98], each editor gets a full copy of the files or assets. Unfortunately, when they attempt to reconcile their changes with another revision, they might find that there are conflicts and these have to be merged.

2. The second concept that is useful is whether the merging support is *state-based*, *change-based* or *operation-based*. State-based merging uses only the state of the files or assets at the time of merging. It is characterised by the standard visual differencing tools that are common to modern development environments. In contrast, change-based merging considers all individual changes done to a version of a file or an asset. A specific flavour of this, operation-based merging, associates each change with a particular operation in an editor. With operation-based merging, it may be that the operations can be re-run inside the editor, and two or more sets of operations can be interleaved appropriately to achieve a satisfactory merge. This can be supported as an extension of action tracking in modern editors, and it is the basis of a relatively recent work on version control of 2D images [CWC11]. This thesis, however, focuses on a state-based merging, see Chapter 4, because common 3D formats do not easily identify operations, and because the system needs to support a broad range of authoring tools, see Chapter 3 for the reasoning and motivation.

3. Next is the distinction between a *raw*, a *2-way*, and a *3-way* differencing and merging. In a raw merge, which is supported by change-based systems, if two files conflict, the merge is effected by transferring all changes from the second into the first file. This is prone to problems, and thus needs verification [ASW09]. In a 2-way differencing, two versions of one or more files, typically a single file in most common tools, are presented and the editor selects changes from both to create a new version that combines modifications or parts from both input versions. This type of tool is commonly provided in modern IDEs. A typical layout would include side-by-side views of two source code files with highlighted additions, deletions and modifications. In a 3-way differencing, the two conflicted versions are compared against their common origin. As further discussed in Chapter 4, in a 2-way differencing, if a change is in one file but not the other, it is unlikely that it is possible to tell whether the change is an addition from one file, or a deletion from the other. With the addition of their origin, there is no more ambiguity since the common ancestor is known.

4. Finally, there is the distinction between *textual*, *syntactic*, *semantic* and *structural* changes [Men02]. Most differencing tools use just textual changes, that is, they do a line-by-line change detection [HVT98] based on implementation similar to the original file differential algorithm, [HM76].

Line-by-line changes work reasonably well for code and other text-based files, but they do not work well for highly structured or binary data. For code, syntactic and semantic change detection tools can identify the language being edited. Structural changes are common in code situations, when a class is refactored, or a piece of code is isolated as a function. There, it can be that these are extensive textual but little functional changes. The 3D Diff tool [DS12a, DS12b], described in detail in Chapter 4, performs a form of semantic change detection because its focus is on a scene graph representation of 3D models. See also a related data organisation discussion in Section 3.2.

2.3.1 Visual differencing

The previous section already described differencing tools designed specifically for text files. Most modern IDEs include some form of visual differencing, e.g. FileMerge for XCode [Wen14] or WinDiff for Visual Studio [RGAM10]. These tools allow the user to visualise the differences and also create a new file that integrates changes from revisions that are being compared. Because of the problems identified with textual changes not representing syntactic, semantic or structural properties of the original files, it is common amongst such tools to allow raw text editing for situations where a coherent version of the file cannot be formed simply by selecting the sets of conflicting edits from either of the input files. To support structured data, there have been many extensions to other domains, where the differencing tool models the contents of the files, rather than the file itself. Firstly, the tool must be able to detect conflicts in the model and then visualise them appropriately. Secondly, the tool might, but will not necessarily, allow some form of editing.

The ability to detect and highlight differences falls within the information visualisation and human-computer interaction (HCI) domains. A comprehensive survey by Gleicher et al. [GAW+11] identified three main strategies for visualising differences: *juxtaposition* (side-by-side), *superposition* (overlay) and *explicit encodings* (time warp or subtractions). Juxtaposition visualisation is the most obvious form characterised by text-based differencing tools. Superposition is known to be superior in the cases where there is more structured data to process such as is the case of trees [MGT+03]. Finally, explicit encodings annotate a diagram with changes. As discussed in Section 4.3, a side-by-side visualisation is the most intuitive comparison technique for 3D data especially due to the difficulty and clutter of superposition or explicit encodings in 3D space. There are useful guidelines for developing these kinds of visualisations such as providing a shared focus and context [WBWK00] that have been applied in Chapter 4.

In software visualisation, especially software diagrams, the problem of change visualisation and merging is well studied, see [FW07] for a review. Since a common data structure to describe polygonal 3D models across various applications is a scene graph [ZHC+00, BRDA11], there is also a connection to graph visualisation tools [MGT+03, HvW08]. The flexible tree matching algorithm [KTA+11], for example, finds corresponding tree nodes even if they have varying descendants. This and similar solutions, however, focus on the structure of the graph while this thesis targets polygonal 3D models with visual differencing as the main user interface. There are also some parallels with work on conflict resolution for 2D hierarchical diagrams [DS10, ZKS11]. The issues are somewhat different as the types of conflicts that are present in a 2D diagram are different to 3D models. For example, restructuring the

diagram can help with conflict detection, something that would not be immediately applicable to general polygonal 3D models where the layout is usually critical and directly influences the final rendering.

One of the first attempts of dedicated 3D differencing is the abandoned Art Diff for SVN project [Mej09]. This tool loads 3D files for a very basic side-by-side visualisation. However, it does not detect any conflicts nor does it support merging. Similarly, Thingidiff [Pat11] is a simple web-based overlay visualisation for STereoLithography (STL) [Ros88] and Wavefront Object [Inc04] file formats such that additions are highlighted in green and deletions in red, but still without any support for merging.

For differencing in 2D images, the world's largest code host GitHub deployed a web-based user interface (UI) extension [McE11]. This tool offers four visualisation techniques, namely '2-up' as a simple side-by-side visualisation, 'Swipe' as an overlay with a slider unveiling one or the other image, 'Onion Skin' as an overlay with varying opacity, and 'Differencing' displaying only the modified pixels. More recently, they also introduced another extension for visual differencing of STL files [Ska13]. Here, the two input polygonal 3D models are overlaid on top of each other. A binary space partitioning (BSP) tree determines what has been added, shown in green, versus what has been deleted, shown in red. This further provides a simple slider to blend between the two models, but same as before, without any support for conflict resolution or merging.

Although standard line-based source code differencing tools are capable of identifying portions of the original file even if displaced significantly from their previous locations, none of the existing approaches for polygonal 3D models can identify the same geometry when repositioned within the same scene. Furthermore, they do not represent the various types of conflicts as done in 3D Diff, see Chapter 4.

2.3.2 Edit tracking

Since standard merging of binary 3D files, their Extensible Markup Language (XML) or plain text counterparts, see Section 2.4, is not feasible, many solutions rely on real-time aspects of simultaneous collaborative editing. One approach to identifying changes is to track all editing operations that the users perform on the files with the expectation that a record of the operations can be used to inform the merge process [WK96]. A macro, i.e. a collection of commands in a graphical user interface (UI) [KF92], can be regarded as an editing history. An example of this is the implementation of concurrency control based on the selection of objects inside a custom-made 3D editing tool [CO00]. In this system, the VRML format [CBM97] was used for data interchange between individual clients. Another example is the conflict detection and resolution in a range of multimodal 3D applications via individual actions depending on their time stamp in a queue [TPZB08]. To lower the data load, this framework passes only manipulation messages to and from the server instead of the entire 3D scenes. This work was further extended to include dynamic locking synchronisation [TZ09] which also relies on a server-side queue just like [CO00] in order to resolve the sequence of edits and their application to a common 3D asset. A similar solution was implemented as part of the Uni-verse system, see §2.4.3 for details. Proprietary high-end CAD systems also provide comparable functionality. For instance, Dassault Systèmes Enovia [Das12b] enables collaborative editing over the network which can be considered a form of real-time simultaneous editing as the users have to work synchronously. Capvidia's CompareVidia [Cap12] validates and compares

CAD models and classifies them as passed or failed based on user criteria. Similarly, 3D-Diff by 3DSys [3DS12] is a Visual Basic add-on to Dassault Systèmes Catia [Das11] that lists and highlights structural differences in CAD models. Unfortunately, these commercial tools are tied to specific packages and their algorithms are not known. Those that are known rely on binary-level delta changes that despite saving space provide no direct semantic understanding of the scene changes.

Likewise, scientific workflow management systems capture dataflows for visualisation and manipulation by recording individual user actions. VisTrails by University of Utah is an open source system with a broad support for versioning of scientific workflows and visualisation of datasets from their revisions. This system provides multi-view visualisations, graphical provenance trees, i.e. histories, and *undo-as-delete* functionality [BCC+05, DF08]. VisTrails' commercial Provenance Explorer for Autodesk Maya [Vis12] includes a side-by-side as well as an overlay 3D visualisation tool. However, the differences are detected and merged based on their underlying provenance histories stored in a relational database rather than by the extraction directly from 3D files [Cal09].

In the domain of computer graphics, collaborative editing and visualisation, e.g. the generation of photo editing sessions based on author demonstrations [GAL+09], are increasingly becoming important. The Chronicle system [GMF10] visualises the editing history of 2D documents in a timeline, but to do so, it instruments the editor and video records the entire session. This tool supports document history exploration by linking the editing events and components of the UI into playback functionality. The resulting video is indexed and hierarchically clustered what results in large data sizes for generated sequences. This system was extended to support Autodesk's CAD software [Aut12] where 3D modelling sessions are video-recorded and shared via a dedicated website. Similarly, a non-linear versioning system for 2D images [CWC11] based on the GNU Image Manipulation Program (GIMP) [GS10] is operation-based and thus records user actions so that they can be replayed via a revision tree. The system uses a DAG to represent sequences of edit operations as graph nodes and the corresponding spatial, temporal, and semantic relationships as graph edges. These recorded graphs are then visualised to display a revision history. In addition to the side-by-side differencing, it also provides action replay with a varying speed. Its merge UI shows two revision images and a preview of the final combined result such that any conflicting modifications have to be resolved either manually or by selecting one of the revisions. In the 3D domain, the Meshflow system [DKP11] obtains sequences of user actions via an instrumented Blender [Bla12] plug-in which records all editing operations. By clustering the edits in several layers of regular expressions, this system allows for interactive playback of the modelling history. The algorithm analyses the frequency of repeated operations that can be filtered by type or vertices that have been affected. However, it only deals with simple meshes, does not scale to large 3D scenes and provides no support for semantic understanding of changes. Finally, a very recent tool for generating history assisted views from 3D modelling sessions [CGW+14] generates informative view points during the editing process. This system records and subsequently investigates the modelling workflow alongside a few discrete snapshots of the model itself in order to infer important regions for visualisation. In addition, the system can visualise the time the modeller spent on a specific region via a heat map.

Whilst edit tracking as a form of change management is certainly viable, it requires the instrumentation of the editing software and thus is only applicable if it was enabled during the creation process. Hence, the 3D Diff system developed in Chapter 4 is state-based rather than change or operation-based. The reason for this decision is the vast variety of tools that can be used to edit 3D models, see Chapter 6.

2.3.3 Mesh compositing

Combining parts of polygonal 3D models or sections of individual meshes in order to automatically generate novel shapes is a broad research area [SBSCO06, KJS07, Sch10]. Meshmixer [SS10], also a part of Autodesk, offers a user interface for rapid mesh composition. By geometry drag-and-drop and mesh cloning they support a seamless transfer of detail from one mesh to another. A standard technique for establishing analogies between 3D models is to compute a signature for each part [SSCO08] and rely on a contextual part-based hierarchy to add support for certain configurations of components [SSS+10].

Over the years, researchers have investigated how to compute consistent alignments and correspondences within surface pairs [VKZHCO11], and on collections of 3D models [NBCW+11, HKG11] as well as structural regularities within the same scene [PMW+08]. Since in many contexts point-based correspondence can be ambiguous and fuzzy, a more abstracted part-based correspondence has been investigated. Golovinskiy and Funkhouser [GF09] first proposed consistent segmentation in the context of mesh pairs. They relied on rigid alignment and nearest neighbours to establish correspondences, and jointly segment all the input models into separate components. Subsequently, several methods have been developed to address the problem of consistent segmentation and labelling by clustering points in an embedded space of local shape features [KHS10, HFL12, SvKK+11, WAvK+12] or by coupled modal analysis [KBB+13], either in supervised or unsupervised settings. Recently, Kim et al. [KLM+13] jointly optimised for correspondence, part segmentation, and part-level deformation to analyse model collections. Unlike these algorithms that investigate similar objects coming from related shapes, the focus of 3D Timeline [DMS14] is to reverse engineer an editing sequence from multiple frames of a modelling session. Similarly to the recent inverse image editing [HXM+13], it proposes a set of simple geometric rules and methods to extract common edit operations and the semantic provenance. The goal is to detect operations such as mesh refinement, instancing, etc., while the untouched parts of the meshes remain identical across the frames. The linear sequences of input models are not suited for co-analysis, since geometric similarities are confined to neighbouring scenes, see Chapter 6.

In the presence of point-level correspondence information across pairs of meshes, early efforts in computer animation proposed multiresolution mesh interpolation frameworks [LDSS99, MKFC01]. Subsequently, algorithms have been developed to interpolate mesh collections by constructing and navigating underlying shape spaces [SZGP05]. In contrast, a system for part-based 3D recombination [JTRS12] establishes mesh correspondence in two unrelated models in order to synthesise new 3D shapes. Using shape analysis, it determines which parts of the models are in contact and symmetrical. The UI provides a single slider to control the interpolation weight between the models when replacing matching sections. Even though this processing is completely automated, many of the resulting shapes are not desirable or visually satisfying. The 3D Timeline interface in Chapter 6 also interpolates models

via a slider, although the challenge there is to reverse engineer the edit trees from the input data. More recently, in an interesting system MeshGit [DP13], the edit distance between meshes is approximated as a cost of matching elements. By treating faces and vertices as nodes of a graph with edges representing their adjacency relations, it converts the task of mesh differencing into a maximum common subgraph isomorphism problem. The method relies on spatial adjacency and does not handle modelling operations such as instantiation, free-form shape manipulation, remeshing, etc. A comparison with this algorithm is presented in §6.5.2. What many of these works have in common is the idea of generation of 3D shapes and whole models that did not exist before. This can be achieved via consistent segmentation and analysis of the input models. However, these solutions do not determine the actual differences between parts of models, the main requirement of the system presented in Chapter 4. Such techniques might be useful in the future when it can be detected whether explicit conflicts have a similar shape or not, see Chapter 7.

2.4 Asset Distribution

Distribution of 3D assets over the Internet can use different types of networked protocols and data representations. In the HyperText Transfer Protocol (HTTP) [FGM+99], it is the client who requests the most appropriate representation of resources depending on their intended application. For example, there exist several encoding formats for the web pages, e.g. HyperText Markup Language (HTML), Extensible HyperText Markup Language (XHTML), etc., each providing its own set of advantages. For 3D data, the distinction of encodings is even more important as very different component types and file sizes make up the structure of a complex 3D scene. As recently demonstrated by Jung et al. [JLH+13], by relying on a quantisation it is even possible to render a 91 million polygon model in a web browser. It is, therefore, important to examine various data representations in order to understand their benefits and drawbacks that might further influence the design and evaluation of a web enabled system proposed in Chapter 5. Several previous attempts at providing online access to 3D assets have been implemented before. An ongoing *rest3d* initiative sponsored by Advanced Micro Devices, Inc. (AMD) [PA11] proposes to define a REST [Fie00] interface shared by all 3D resources on the web. The only suggested delivery formats are XML [BPSM+08] and JSON [ECM13]. However, both are considered outside the scope of any version control. Furthermore, the benefits of a REST web service integration have been demonstrated on a C++ scene graph system OpenSG [SBU+10].

Historically, the first standardisation efforts began in 1995, setting out the requirements for a scene description language for the web [LHM97]. This lead to the adoption of a proposal for a static scene definition based on the Open Inventor [Wer94] API and file format referred to as the Virtual Reality Modeling Language (VRML) [Rag94]. The initial version 1.0 was a simple graphics definition made up of transformation and geometry nodes forming a tree hierarchy commonly known as a scene graph, see Section 3.2. Just like HTML, VRML supports the linking and embedding of other media types including 2D and 3D documents. Later, the static definition was extended to support interactive scenes and sharing via new node types for sound, video and mechanisms to associate scripts, animations and events with 3D objects [HMRL95]. These extensions were combined into a proposal for VRML 2.0 which became an ISO standard known as VRML97 [CBM97]. This added a routing mechanism that enables events

to be generated and passed onto sections of a scene graph such as script nodes. With the continuous advances in 3D graphics, the VRML Consortium was renamed to Web3D Consortium which introduced a backwards compatible successor to the standard called X3D [JIS+13] as well as an integration model for HTML5 called X3DOM [BEJZ09]. This lead to the introduction of experimental binary container nodes such as ImageGeometry and BinaryGeometry [BJFS12] as well as POP Buffers [LJBA13]. These concepts are now being transformed into a Shape Resource Container (SRC) [LTBF14] as a candidate for standardisation in X3D 4.0. A competing declarative 3D for the web is Extensible Markup Language 3D (XML3D) [SKR+10] with its dataflow extension Xflow [KSR+12] and an equivalent binary encoding format Blast [SSS14]. A prototype server supporting `POST` and `GET` methods with XML3D rendering, although without external referencing of resources, was devised previously [SH12]. Chapter 5 follows in these footsteps and defines a fully specified API with several example data encoding implementations where significant considerations were made for the speed of data delivery on various platforms including mobile devices as well as versioning. For differences between X3DOM and XML3D, see Section 5.3.

A recent Massachusetts Institute of Technology (MIT) spin-off, Sunglass [DR12], provides a proprietary Web Graphics Library (WebGL)-based [Mar11] collaborative 3D modelling solution. Their paid for servers can be accessed via a REST API in order to manage JSON mesh representations alongside the original binary files. Version control in this system stores a linear history of binary snapshots without delta changes, lacking support for differencing or merging. Autodesk's CAD package AutoCAD also supports editing and sharing of CAD drawings via its online viewer and mobile apps suite [Aut14a]. In contrast, the open source visualisation system, VisTrails [BCC+05], provides a broad support for the versioning of scientific workflows. Despite the provenance history being represented as XML or stored in a relational database, the actual 3D files have to be managed locally. The spreadsheet-like workflows can be visualized using the Visualization Toolkit (VTK) [SML06] and HTML renderings. Similarly, a scene graph rendering engine SceneJS [KOL+09], provides a high-level JavaScript API for WebGL. Its JSON-based representation can be easily transported over the network and stored in a DB. On the other hand, repositories such as the former Google Warehouse, now known as Trimble 3D Warehouse, TurboSquid or 3D Repository by the Advanced Distributed Learning (3DR) are large online libraries of 3D assets. Despite offering searchable web interfaces to locate predefined file formats, only 3DR provides a basic REST API accessible using XML and JSON encodings [Adv11].

In summary, most online 3D APIs support the XML and JSON data formats, although many lack version control altogether. There are of course many more formats and approaches to chose from and the taxonomy of 3D data encodings for the web is summarised in Figure 2.3.

2.4.1 Text formats

The 3D data formats for the web can be categorised into text-based and binary formats. In general, formats encoded as text are human-readable, although their file size is usually much larger than of the equivalent binaries, see Table 5.2. They also need to be parsed which causes an undesirable decoding overhead increasing the initialisation time before the rendering takes place. The text encodings can be further subdivided into two distinct groups of document and pure data-based formats as follows:

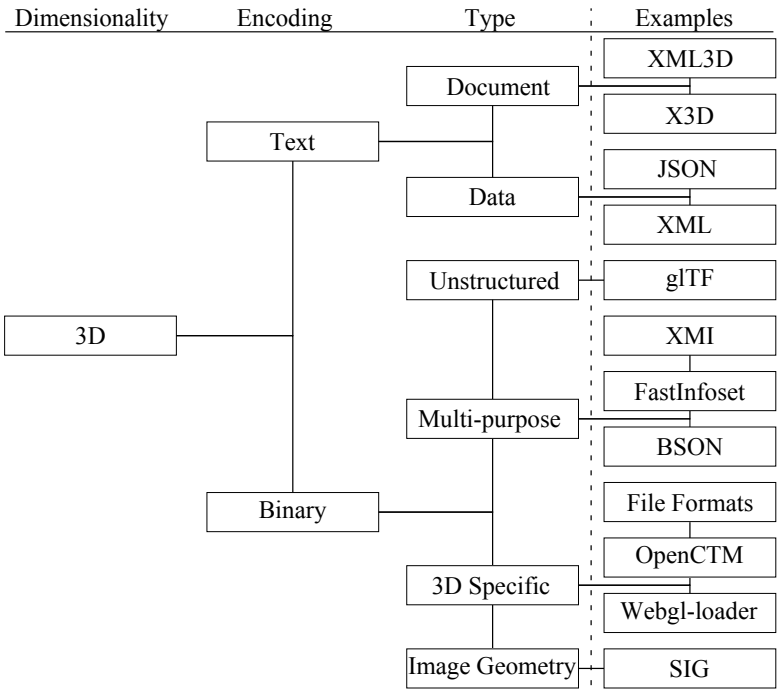


Figure 2.3: *Taxonomy of 3D data representations for the web [DSR+13].*

Document-based. Usually, the document-based formats represent a whole scene and also some runtime information in order to enhance an otherwise static visualisation. A common approach is to group geometry together with animation and shader resources directly within the same document. This makes the resources available to the document object model (DOM) [Mar02] API at the parse-time, even though the resulting string encodings tend to be larger in size and, therefore, take longer to download and render in contrast to binary formats. Often, the interaction with a 3D model is possible only once the parsing of the entire document has finished. Inclusion of resources as an attribute or a character component of XML can be found in popular formats such as COLLADA and X3D. In the context of declarative 3D for the web, a document is simply an HTML page describing one or more 3D scenes. In such a document, the resources are encoded internally or externally using uniform resource identifier (URI) semantics. Alternatively, it is possible to include only those resources that are required at the runtime while referencing all static elements externally, see Figure 5.2 for an example.

Data-based. Multiple formats can be used to encode referenced resources that are external to the document. Following the REST [Fie00] principles, externalised text encodings can contain either a single resource or a collection of resources such as all shaders of a 3D scene in one external file. Modern web browsers provide functionality to load such resources during the runtime. XMLHttpRequest (XHR) [KASS14], the main component of Asynchronous JavaScript and XML (AJAX) [Pau05], defines an API to transfer data between a client and a server. These were already used to enable X3D sever communication, most notably to preserve user annotations in a NoSQL DB [Olb12]. Fortunately, XHR is not restricted to XML only. Together with JSON, these two text-based encodings are the most common external formats used on the web since web browsers expose native parsing capabilities for both. JSON is particularly popular since in contrast to XML, it omits the end tags which makes it smaller for highly

structured documents that have small data entries. For polygonal 3D models, however, the difference is negligible as these consist mostly of large vertex and normal arrays while the structure accounts for only about 5% of the overall scene size [BJFS12]. See Table 5.2 for comparison of XML and JSON file sizes. Nevertheless, JSON does not offer a natural way of addressing its internal elements. Web browsers, on the other hand, provide cascading style sheets (CSS) Selectors and XML Path Language (XPath) to access elements of DOM, the in-memory representation of an XML document. In HTML, the uniform resource locator (URL) fragment can be used to refer to elements by their identifier (ID) attribute. Hence, XML, unlike JSON, can be used to represent collections rather than just the individual resources. Both formats can be compressed with the Deflate [Deu96a] or Gzip [Deu96b] compressions that are available in all major web browsers. This increases the decoding time but reduces the bandwidth requirements when transmitting the data over the Internet. Despite this, they both suffer from the issues of any generic string representations. For the WebGL API to be able to access such data, the strings have to be deserialised into *Typed Arrays* [Khr13].

2.4.2 Binary formats

Binary encodings are often used to represent proprietary data formats. Some of these can be uploaded to the GPU directly to avoid unnecessary parsing overheads in JavaScript. As shown in Figure 2.3, they can be further subdivided into four categories as follows:

Unstructured buffers. Binary data can be transmitted via XHR as *ArrayBuffers* [Khr13] that represent any generic fixed-length raw binary data buffer. These cannot be manipulated directly, however, it is possible to generate Typed Arrays or a specific data view from them. For example, every four bytes can be treated as one float entry thus deriving a `Float32Array`. This strategy is used for an internal data storage of a newly proposed 3D version control repository as demonstrated in §3.4.2. Obviously, such an encoding provides no explicit structure, hence for 3D data transmission one approach is to request a buffer per vertex attribute as proposed in BinaryGeometry [BJFS12] or glTF [RPO12]. Unfortunately, for very large scenes this results in a large number of XHR requests that leads to a reduction in performance especially over high-latency connections. With many concurrent users this would cause the server to become unresponsive. However, multiple vertex attributes can be interleaved into a single ArrayBuffer.

Multi-purpose. General structured data can be encoded using multi-purpose binary formats. Several competing standards for binary XML exist, for instance XML Metadata Interchange (XMI) [Obj11] and FastInfoset (FI) [Tel05]. A common approach is to rely on a dictionary compression for element and attribute names and a binary encoding for actual XML data types that can be further compressed, e.g. using the Deflate algorithm as in XMI. Binary encoding of X3D is based on FI in order to exploit its capabilities such as referencing predefined dictionaries and custom compression methods. These make the FI a hybrid between a generic XML encoding and a domain-specific compressed format that can achieve very high compression rates within a generic format [SS11]. However, no readily available JavaScript implementation of FI exists so it is not yet utilised on the web. On the other hand, a binary derivative of JSON is BSON [Mon14a], that was originally developed as an internal data format for MongoDB [MPH10]. This encoding does not support any dictionary compression but provides the

means of representing the structure and data types in a binary document that can be easily deserialised during the runtime. In addition, BSON contains extensions for data types that are not part of the standard JSON specification such as a byte array or a date, while omitting its universal number type.

3D specific. Many open and proprietary domain-specific 3D file formats also exist. Some of these exploit the knowledge about the data properties for the purposes of compression. Open Compressed Triangle Mesh (OpenCTM) [Gee09] and WebGL-loader [Chu13] are of particular interest for web 3D data delivery because they come with a JavaScript decoder. Both use classical compression schemas such as Delta [HGF+02] and ZigZag [Goo14b] encodings. In addition, the WebGL-loader exploits the variable-length encoding of Universal Character Set Transformation Format–8-bit (UTF-8) [Yer96] to capture any values with one to three bytes. On the web, any structured binary formats need to be decoded from their binary representation into JavaScript since they are not directly compatible with Typed Arrays. The time required for the decoding is significant, especially on mobile devices. To prevent blocking of the UI in the web browser, it is possible to move the decoding into a separate threat using a web worker [Pil10].

Image geometry. A special case of a binary 3D format is encoding geometry in images since there is no requirement to modify the data in JavaScript. Images are treated simply as data vessels that carry bytes. These are decoded by the browser natively and can be uploaded to the GPU directly to serve as a data buffer. Sequential Image Geometry (SIG) [BJFS12] extends this idea even further by splitting vertex arrays into 8-bit chunks of decreasing relevance that are distributed as a sequence of images. By omitting the later images, i.e. the least significant bits, the approach supports quantisation and progressive loading as long as the images arrive in the correct order. However, SIG also requires data fetching from images in a vertex shader that is not yet supported on many mobile devices. On those devices that support four texture units, these would be occupied by vertex coordinates with two images representing 16-bits, one image for eight-bit normals and one for eight-bit texture coordinates. If none or too few texture units are available, JavaScript can create Typed Arrays from the images, although this would degrade the performance. Nevertheless, images were never designed to represent 3D data, so formats such as Portable Network Graphics (PNG) [Bou97] cannot achieve high compression rates for such assets [BJFS12].

2.4.3 Networked protocols

Related to storage and control of assets is the way in which they are transferred over the network. Many forms of distributed virtual worlds and games also provide some sort of networked 3D formats [SO09]. A database implicitly supplies a query language and this can usually be accessed over the network. Of particular interest are the systems that support distribution of complete scene descriptions alongside scene edits. Systems such as Distributed Open Inventor [HSFP99] supported the distribution of a full scene graph and subsequent real-time edits. A related concept, the use of a scene graph as a data format to intermediate between different applications, was presented in the scene graph as a bus project [ZHC+00]. There are of course many remote desktop protocols such as the Remote Framebuffer protocol [RW98], Windows Remote Desktop Protocol (RDP) [Mic14], Google Chromoting protocol [Goo14a], etc., but these transfer desktops through frame buffer capture rather than actual 3D content. A successor to RDP is RemoteFX [Mic11] which provides a 3D virtual adapter, codecs and other functionality.

Second Life, described in detail in §2.1.1, relies on a standardised open protocol for sending and receiving 3D models as serialised XML via a REST API [Len08]. In principle, this protocol could be re-purposed for sharing of any kind of 3D assets over the Internet, but in practice it is customised for a real-time application and not for a general-purpose data retrieval. Additionally, it is highly constrained to the types of data encodings by the platform it was designed for, hence it is not as flexible as other web 3D data representations such as XML3D.

In contrast, a somewhat similar approach to the CVEs from Section 2.1 was taken by the Uni-Verse project [BS07] which was a collaboration between The Royal Institute of Technology (KTH), Helsinki University of Technology (HUT) and the Blender Foundation until 2007. This open source online platform supported sharing of 3D geometry, high-dynamic-range imaging (HDRI) textures and shaders, all distributed from a centralised server to the connected clients. The main system was built around a low latency network protocol *Verse* on top of which different clients and plug-ins for modelling tools such as Blender [Bla12] were created. All local changes were automatically distributed to the server and pushed further down to the remaining clients. Unfortunately, the integration of Verse protocol was dropped from the main trunk of Blender in version 2.5.

The Verse 2.0 protocol [Hní10, Hní12], introduced during the 9th annual Blender conference, is a revival attempt addressing some of the deficiencies of the first version such as weak security and utilisation of only a single socket. This further introduced a new packet structure listing version number, `ACK` & `NAK` flags, system and node commands as well as locks granting access permissions to individual assets that is a significantly more complex definition than the original version. Nevertheless, in [Hní11], it was shown to provide better performance than any one of the User Datagram Protocol (UDP) [Pos80], Transmission Control Protocol (TCP) [Pos81], Scalable TCP (STCP) [Kel03] and Datagram Congestion Control Protocol (DCCP) [FHK06]. This system on its own, however, does not provide any persistent state preservation so as soon as the server is switched off or rebooted, the already created 3D models and user interactions are lost. Recently, the ability to store the final state of the server in MongoDB [MPH10] before exiting was proposed [Hní13]. The current development is primarily focused on reliable congestion control, prioritisation and scheduling of data to be sent to the client. This will increase the probability of successful delivery of high priority data blocks.

Each of these network protocols demonstrates that changes can be propagated but this is only one aspect of a distributed 3D editing. Nevertheless, such protocols could be useful in the future when the bandwidth requirements of the database transfers become limiting due to the sheer volume of data or the number of connected users, neither of which fall within the scope of this thesis.

2.4.4 Gaming on demand

A separate category in the distribution of 3D content has been recently created by the Gaming on Demand (GoD) services that offer instantaneous access to high-end game play and even whole operating system (OS) over the Internet. Solutions such as OnLive [OnL09], Gaikai [Gai12], Otoy [Hen08], etc. render game content on a server and transfer the scenes to the end user as a compressed video stream. In exchange, the users continuously supply commands that control the navigation enabling nearly identical

level of interaction as experienced when running the game locally. However, the real-time 3D rendering and interaction significantly depend on the users' bandwidth and latency to the server [CWSR12], which currently limit access to these services to a close proximity to the cloud only [CCT+11, MHUC12].

For OnLive, the accessibility range used to be artificially limited to a fixed radius from the data centre [Shr10]. Apart from a dedicated desktop interface, OnLive also offers its own console with a remote controller, similar to standard gaming consoles that are directly connected to a television set. Within this console, all of the video decompression and game navigation happens on an underclocked dual-core 1.2 GHz Marvell Armada 1,000 chip with 512 MB random access memory (RAM) [Hol10, Ste10]. OnLive was also the first to present a demonstration of Autodesk Maya 3D editing on Apple iPad executed on Microsoft Windows 7 [Mar10], despite neither of these being supported by the tablet hardware natively. At the same time, a Samsung Galaxy Tab was able to spectate the live edits that were streamed from a datacentre 50 miles away. After the financial collapse and restructuralisation of OnLive in 2012, this functionality is now being offered as the OnLive Enterprise solution. Nevertheless, OnLive was shown to suffer from a "downstream turbulence", i.e. an uneven distribution of packets sent to the client [CFGS12]. This is akin to a high-definition video due to utilising large packets and high bit rates.

In contrast, Gaikai, acquired by Sony Computer Entertainment in 2012 [Gai12], is currently being integrated into Remote Play for PlayStation 4 (PS4). At the very first public testing session by Engadget [Hol12], with a distance of roughly 15 miles to the cloud and latency between 25 to 35 ms, the reviewers experienced a significant input lag while receiving 720p video stream. Reported were also compression artefacts mainly visible around small text with less smoothing when compared to OnLive. Furthermore, in 2013, AMD unveiled Radeon Sky Graphics cards [Adv13] that power their streaming cloud solution similar to other GoD services. It is generally believed that such solutions require at least a single processing unit per user making them computationally demanding [MSG11]. A similar service is provided by RealityServer [Mig14] originally developed by mental images. This cloud-based rendering provides real-time ray tracing using Iray [NVI09] that is streamed to the clients.

These proprietary systems inspired open development such as the University of Southern California (USC) Game Cloud [ZHV12], GamingAnywhere [HHCC13], a low-end architecture [BLCR10], etc. Even though it would be possible to connect a version control system from Chapter 3 to a cloud-based rendering system for visualisation purposes, the end-users would still need full access to the assets in order to modify them locally. Such a requirement can be overcome if the entire modelling runs on the server in which case the visualisation is not much different from a remote desktop session. In addition, the service provider has to maintain data centres scattered around the world, increasing the costs and environmental impact of such solutions. On the other hand, the owner has a full control over the content and its delivery, reducing the problems such as software piracy, etc. Although the main benefit of a thin client solution is its portability and low end-user hardware requirements, there are still the problems with off-line accessibility as well as scalability to many simultaneous users [CFGS12]. For instance, the extremely high running costs are quoted as the main reason for the collapse of OnLive [Par12].

2.4.5 3D maps

Online 3D maps are one of the earliest web browser-based 3D experiences available to the masses. Google Maps [GE06] is an online service offering Mercator projections [Mal92] of political and satellite views of the whole world as well as of the street view which provides 360-degree panoramic images at the street level from various locations. In addition, the map interface at a close range displays basic 3D shapes of buildings. Data in these services is transferred as JSON rather than XML due to its compactness. The available API is popular with the GIS community as it allows custom extensions such as a visualisation of London [GSM+08] to be built on top. *MapsGL* is a WebGL version of Google Maps with hardware accelerated transitions and overall smoother performance [McC11]. To support wide-spread content creation for its maps as well as the Google Earth software, Google has developed several online applications for 3D modelling. Google Building Maker used to be an online 3D editor that enabled its users to create basic models from aerial photography so that they could be submitted to Google Earth. This application was, however, discontinued in 2013 [HÏ3]. Instead, the Google Map Maker [Goo08a] is meant to replace the demand for future 3D content creation in Google's services. This interface, similarly to Building Maker, relies on quadrilaterals to approximate the shape of buildings from aerial imagery. Bing Maps [Mic10], Microsoft's version of online maps, offer many similar features including street level imagery and aerial photography. In addition, Bing Maps used to provide texture-mapped 3D models of some of the world's largest cities with the ability to add custom models using 3DVIA Shape for Maps software [Das12a]. Similarly, Here [Nok13], formerly known as Ovi Maps and later renamed to Nokia Maps, is a mapping service that, on top of the standard set of features, provides 3D views of several major cities as well as an earth-like rendering equivalent to Google Earth but in a web browser. 3D buildings in this system are automatically generated and texture mapped using a combination of aerial photography and laser, hence the low polygon count of the actual models. Unlike the proprietary services, OpenStreetMap (OSM) [Coa09] follows in the footsteps of Wikipedia and establishes a new approach to collaborative map content creation in order to develop a freely accessible spatial database. Its web-based display relies on OpenLayers JavaScript library [Met06] which embeds dynamic maps in web pages. The OSM-3D [OSNZ10] shows that it is possible to automatically generate web-based 3D city models from such a database with the addition of height information provided by the Shuttle Radar Topography Mission [FK00]. Buildings in this system are once again interpolated quadrilaterals with flat roofs. This website is maintained by University of Heidelberg and offers a 3D view of the data stored in OSM. Their solution is based on a scene graph with elevation model, although the viewer needs to rely on a Java plug-in and hence is not as portable as the alternative solutions. [GZ12] further show that it is possible to use this voluntarily produced data to generate building models and more recently even analyse the topology and geometry of street networks [AYWM14]. OSM Buildings [Mar13] is a JavaScript library for visualising 3D buildings in this system that relies on a 2D canvas element of HTML and does not require any WebGL support. There is also an XML3D viewer [Joc12] for the OpenStreetMap and the OSM Buildings. Despite being collaborative in nature, these systems were not designed for large-scale development of engineering 3D models as required in Chapter 1. See Figure 5.4 for an example.

2.5 Chapter Summary

This chapter has been divided into four main sections that cover the key relevant concepts from academia as well as industry. With the ever increasing diversity of 3D authoring tools, the skills required to operate them and the geographical spread of individuals across the globe, virtual collaboration is becoming more important than ever before. Section 2.1 introduced the prominence of collaboration in large-scale industrial projects. Despite the research into collaborative virtual environments (CVEs) spanning nearly three decades, these systems are not directly applicable to the development and distribution of complex 3D models. Although the idea of a Metaverse, i.e. the convergence of physical space and the Internet has been around since the early nineties, it was only recently that the game-like environments such as SecondLife and Minecraft witnessed a surge in the general interest in collaborative modelling. These systems, however, suffer from general low fidelity of 3D models and the lack of support for advanced editing tools. In the case of Minecraft, the entire virtual world is based on pixelated cubes only. Still, its collaborative nature inspired a worldwide craze with entire countries being recreated in 3D.

Section 2.2 focused on the aspects of digital asset management and the advantages of databases (DBs) over file systems (FSs) when dealing with large-scale 3D modelling. Many approaches to the virtual collaboration rely on some kind of searchable persistent storage. Due to the vast number of software vendors and domain-specific editing solutions, most of which are proprietary, interoperability of 3D data is achieved via an exchange of files on a file system. Although many dedicated data management solutions exist, they are often bound to specific software packages and file formats, or deal with 3D models at the level of individual files rather than actual scene components or changes. The problems that arise from such an arrangement include version control, asset tracking and access control. In contrast, relational DBs have been extensively studied as a suitable alternative to file systems for the purposes of CAD management and geographical data representations. However, these are not suitable for general 3D modelling and edit tracking that would fit equally well into the existing production pipelines of creative industries. Fortunately, the recent NoSQL movement promises flexibility and scalability developed especially with the modern web applications in mind. Since most polygonal 3D models are organised in hierarchies of objects or engineering assemblies, Chapter 3 examines the suitability of document-oriented NoSQL DBs for the purposes of version control in these kinds of 3D models.

Section 2.3 further explored the lessons learnt from the fields of software engineering and more specifically source code version control that might be transferable to computer graphics. There are already many VCSs with extensive tool support and full IDE integrations available. Thus, important concepts from software engineering might be useful for management of 3D assets. Firstly, there is the difference between pessimistic and optimistic version control which determines the need for locking or merging. Secondly, there is the distinction between state-based, change-based or operation-based merging depending on what kind of data informs the process. Thirdly, there are the concepts of raw, 2-way and 3-way differencing depending on the type of VCS and the presence of the common origin of the differenced models. Finally, there is the difference between textual, syntactic or semantic change detection. Some of these concepts have been incorporated into the visual 3D Diff tool presented in Chapter 4.

Although a common approach to collaborative 3D version control is to instrument the authoring tools in order to record all the editing operations, this thesis focuses on state-based merging. Whilst the aim is to preserve the intentions of users in their editing operations, the system also needs to deal with the wide range of editing tools that are used in 3D modelling. Thus, this thesis does not explore the route of software instrumentation and instead concentrates on importing and exporting files into a DB, see Section 3.1. Edit tracking is an avenue for future work once the plug-ins that connect directly to the DB are developed. Nevertheless, an alternative approach to edit tracking is to reverse engineer correspondence across multiple 3D models and recompose them into a new combined result. Such an approach might be used to effect a merge by combining concurrent edits or, as explored in Chapter 6, to generate a high-level overview of the editing provenance from consecutive 3D files.

Finally, Section 2.4 explores the area of 3D asset distribution including various networked protocols, gaming on demand and 3D map services. Although not designed for 3D data, the HTTP protocol is commonly used for asset transmission over the Internet. Data encodings in formats such as XML and JSON are becoming more and more popular due to their native parsing capability in modern web browsers. Many existing web-accessible 3D repositories provide the export functionality in both of these formats. However, those repositories that also offer REST APIs do not necessarily utilise all of the aspects of this style of architecture such as version control. In addition, there are many more 3D data formats, some of which are compressed or binary, that are also available for the web. Figure 2.3 shows a taxonomy. These different formats are, therefore, evaluated for their efficiency and speed of delivery in Chapter 5. In order to exploit the version control database introduced and developed in Chapter 3, Section 5.2 further proposes a novel REST API that provides the desired flexibility so that the various encoding strategies can be supported.

Chapter 3

3D Revision Control Database

The first research question in Chapter 1 asks whether it is possible achieve asynchronous collaborative three dimensional (3D) editing that scales up to useful model sizes. More specifically, it attempts to determine whether a document-oriented database (DB) can substitute traditional version control system (VCS) systems. This chapter, therefore, presents a novel architecture and a prototype implementation of non-linear revision control framework designed specifically for 3D assets called 3D Repository (3D Repo). This provides VCS functionality but built around a NoSQL DB to avoid their file-based constrains.

The main motivation for this undertaking are the drawbacks of storing polygonal 3D models as files on a file system. Each user loads a particular scene into a modelling tool, modifies it and then re-saves the whole file again, making the unit of access to the scene the file itself. If several users want to edit a section of a scene concurrently, they have to access the same files but without extra infrastructure, managing such changes becomes infeasible. The machine memory is a strict limit, too, as massive industrial models often exceed the capabilities of even the latest rendering hardware. Besides, deciding how a large scene should be partitioned into files is a non-trivial task, too. Splitting meshes across different files causes problems not limited to the management of edits that involve multiple objects and keeping them synchronised. Although version control and asset management systems suffer from these problems, they are commonly used for 3D data administration. A variety of existing open source and proprietary solutions have been documented in Chapter 2. These either do not track fine-grained component-level changes or are bound to a single file format.

Hence, there is the need for a centralised yet flexible framework that would enable transparent asset management and would support multiple users with distributed access. The argument is that the framework should not, initially, rely on any one modelling tool or a specification, but instead, it should support files that are external to the editor. Section 3.4 demonstrates the feasibility of the proposed architecture on examples with up to several million polygons in size via three client applications. These clients support distributed editing over the Internet as well as real-time visualisations using OpenGL for Embedded Systems (OpenGL ES) [ML10] in mobile devices and even web browsers. The main contribution of this chapter, therefore, lies in the unification of design and version control of 3D assets. Given the type of database that is being utilised, the potential long-term benefits include massive scalability, ability to fit into existing production pipelines and even crowdsourced 3D modelling in the near future.

3.1 System Overview

Nowadays, much 3D content is procedurally generated or manipulated using custom and often incompatible proprietary tools. Even collaborative editing of scenes produced by a standard set of modelling packages is not as straightforward as might be desired by the industry. According to Arup, in the case of the King's Cross station in London redevelopment, over 800 engineers with different skills and toolsets participated in this £400 m project that lasted seven years from start to finish [Ove12]. The workload was further complicated by the fact that this Grade I-listed station, opened in 1852, was bombarded during two World Wars. Yet, the cost of a laser scan survey of the entire site would have been prohibitively expensive, hence was never done. By way of contrast, the Linux kernel has been modified by more than 3,700 contributors for free over the past two years alone [Tor14]. Since accurate information and close collaboration are crucial aspects of any such a large project, data exchange is currently one of the major causes of unnecessary complexity and waste in the industry as explained in Chapter 2. 3D Repo addresses these pressing problems by introducing a non-linear management and visualisation framework designed specifically for the most common types of 3D assets. The framework is released under an open source license in hope that the industry will embrace the new collaborative practices of 3D Repo.

3.1.1 Functional requirements

In order to separate 3D modelling from its long-term storage, it is necessary to become agnostic to any specific editing software. Such a system has to load and save various 3D file formats using a framework that stores a unified scene representation in a centralised repository that can be shared by all stakeholders equally. The role of the repository is, therefore, to preserve 3D assets and to provide interfaces and conventions to add, modify and extract *delta changes*, i.e. incremental modifications on them. By supporting standard web-accessible front-ends, it also has to enable connections to modelling packages via intermediary tools, or simply to viewers that visualise selected revisions from the repository. Due to the potentially large data sizes, the system further needs to support retrieval of any full or partial revision that can be exported as a new 3D file. These requirements are addressed in Section 3.3 and Section 3.4.

Nevertheless, a simple linear 3D versioning system, even if file-based, is only marginally better than the *undo* functionality [LAM76] found in most editing packages. Generally speaking, the undo allows the user to remove at least the immediately preceding step, although these days it is more common for programs to track a last in, first out (LIFO) list of multiple steps. Unfortunately, such a list is neither exhaustive nor is it preserved beyond the current editing session unless the file format supports it natively, e.g. Autodesk Maya [Pal13]. A linear 3D VCS such as Sunglass [DR12] further adds the ability to retrieve any revision regardless of its order of creation. Despite several users being able to contribute changes to such a system, if no conflict resolution interface exists, they are unable to work concurrently. Since by definition no branching is supported in a linear VCS, the users cannot work on separate sections of a 3D scene independently of each other causing explicit problems in terms of access control and locking. For these reasons, a truly collaborative 3D VCS has to support non-linear branching and, more importantly, merging as well as provide visual 3D differencing that would highlight incompatible edits and offer merge suggestions akin to the text-based *Diff*. This need is addressed in Chapter 4.

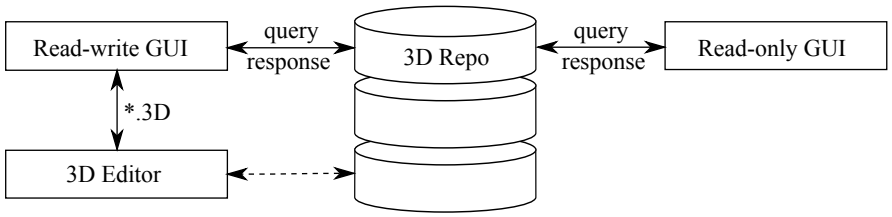


Figure 3.1: *Conceptual framework overview. Editing software exports a 3D file into a GUI that facilitates version control by storing decomposed scenes inside a remote 3D Repository. Lightweight clients further visualise the contents of the repository without the need for write privileges.*

Requirements summary. Summarised below are the key functional requirements derived from formal sessions with Arup Foresight, Research & Innovation, Arup Associates and Balfour Beatty:

1. Store a wide variety of 3D assets including geometry, transformations, materials, textures, etc., independent of the modelling tools that created them.
2. Support domain-specific metadata such as engineering attributes, assemblies or hierarchies.
3. Preserve a traceable audit trail of modifications and manage changes alongside a non-linear revision history in a centralised repository.
4. Enable easy access control and full querying potential of a DB including sub-object retrieval.
5. Implement an interactive user-driven approach to 3D differencing and conflict resolution.
6. Access the repository via lightweight clients without the need for plug-ins or firewall exceptions.
7. Fit into existing production pipelines of architectural, engineering and creative industries.

3.1.2 System architecture

In search of a suitable solution to the problem of 3D version control, recent developments in the No Structured Query Language (NoSQL) DB technology are being exploited here. NoSQL databases avoid rigid table structures and tend to be optimised for large read-write operations while adhering to the *semantic web* [BLHL+01] paradigm. This advocates the inclusion of high-level meaning in web accessible content so that information can be shared and reused across applications automatically without the need for human intervention, although, according to Shadbolt et al. [SHBL06], the potential is still unrealised.

Even though unstructured databases do not necessarily support hierarchical data representations, their ability to organise and query collections of polymorphic documents that are independent of one another make them remarkably suitable to the diverse nature of 3D assets. It is, thus, possible to store entire deconstructed scenes in a DB and, due to its flexibility, also track associated metadata such as semantic relationships of respective scene graph components, their engineering attributes, assemblies, and even revisions. Once in a database, the access is implicitly supported in a distributed manner via a dedicated query language without the need for any functional changes in the DB implementation itself.

3D Repo system uses one of the most popular open source NoSQL databases MongoDB [MPH10] for its centralised data store as outlined in the architecture overview in Figure 3.1. This particular database was chosen because it is proven to scale massively [Cho11, DGG+13] and is built around the Binary JSON (BSON) [Mon14a] specification which, as explained in §3.4.2, is suitable for efficient bi-

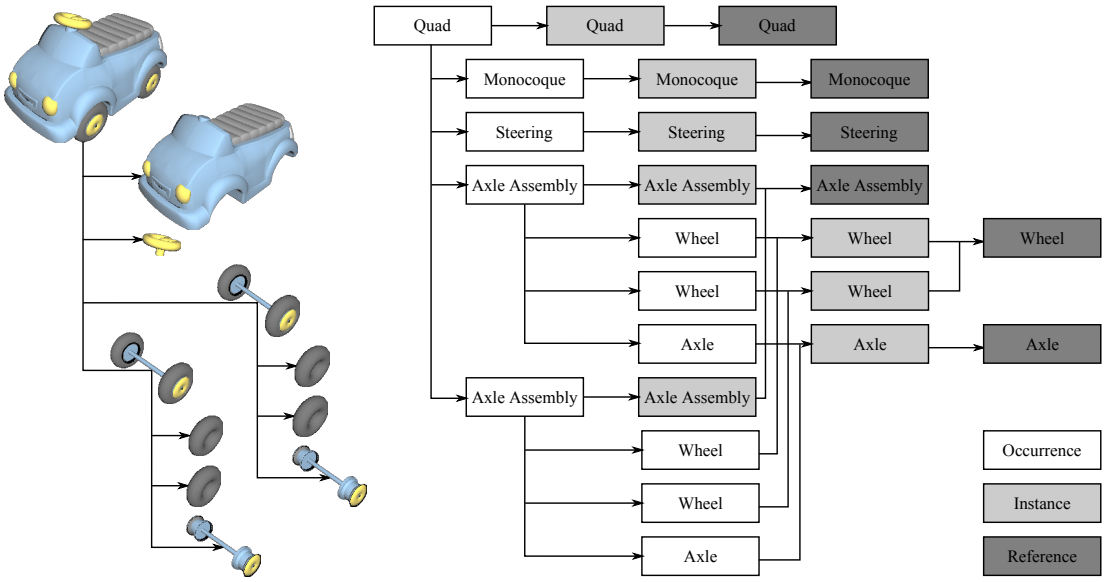
nary storage of vertex-based 3D data. The primary desktop client *3D Repo GUI*, described in Section 3.4, facilitates repository visualisation as well as revision control including *branching* and *merging*. This application became the basis for the development of additional functionality such as visual 3D conflict resolution, *3D Diff*, and reverse engineering of editing provenance, *3D Timeline*, presented in Chapters 4 and 6 respectively. A secondary web-browser-based client, §3.4.4, renders selected revisions using a combination of Java [Sch14], JavaScript [Fla11] and Web Graphics Library (WebGL) [Mar11]. Finally, a dedicated *3D Repo Android App*, §3.4.5, enhances mobile collaboration and feedback collection throughout *public inquiry*, a process that is often required by law, see Chapter 1. These applications are in stark contrast with *XML3DRepo*, a daemon service described in detail in Chapter 5, that provides a layer of DB indirection on the server-side by combining the Extensible Markup Language 3D (XML3D) [SKR+10] and 3D Repo technologies. Although it would also be possible to provide plug-in interfaces to popular modelling packages, shown in dashed line in Figure 3.1, this is outside the scope of the thesis.

3.2 Data Organisation

Geographic Information Systems (GISs) such as Oracle Spatial [KGB11] or PostGIS [OH11], described in Chapter 2, offer the flexibility for generic spatial queries but also consume a significant amount of storage reducing the performance of the underlying database. In contrast, 3D files group every scene component into binary or plain text representations that do not support sub-object queries and require whole files to be loaded into modelling software for editing. Even though many popular 3D datasets, e.g. [SMKF04, CGF09], embody manifold surfaces, examples of large 3D scenes such as architectural models and game levels tend to consist of numerous disjoint components organised in some kind of a hierarchical structure. 3D Repo exploits this natural partitioning as well as the conceptual similarities between a *scene graph* and a *revision history* to define a novel system for non-linear management and visualisation of 3D models. This section, therefore, defines how a scene graph representation of 3D assets can be stored inside a document-oriented DB and how the same DB can be extended to store their associated revisions. Similarly to traditional file-based version control systems such as Concurrent Versions System (CVS) [Ves03] or Apache Subversion (SVN) [PCSF08], 3D Repo preserves delta increments rather than the entire files. Nonetheless, the overarching architecture is independent of the prototype implementation in Section 3.4 and is applicable to any linear data store, not just NoSQL databases.

3.2.1 Scene graph

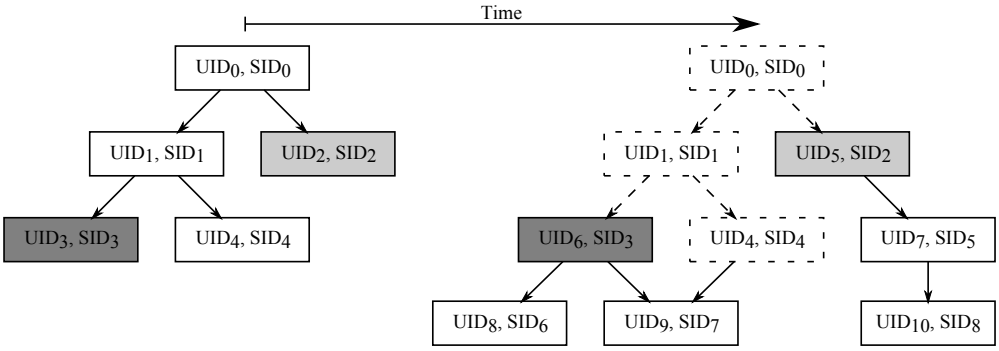
A scene graph is a versatile data structure commonly used to organise, edit and render hierarchical visual information, see Figure 3.2. The Programmer's Hierarchical Interactive Graphics System (PHIGS) [SBM86] was the first standardised scene graph specification which was eventually transformed into Open Graphics Library (OpenGL) [SA94] as well as systems such as Performer [RH94] and Open Inventor [Wer94]. Throughout the history, scene graphs were used to define two dimensional (2D) Scalable Vector Graphics (SVG) [FJJ00] and became the basis for numerous 3D libraries such as the 3D Toolkit [SC92], OpenSG [Rei02], OpenSceneGraph [BO04] and the Visualization Toolkit (VTK) [SML06]. Zeleznik et al. [ZHC+00] and Berthelot et al. [BRDA11] used scene graphs



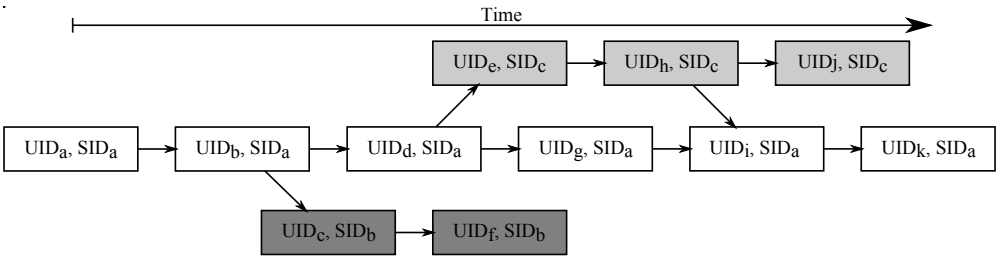
(a) *Modelling scene graph* **(b)** *Rendering scene graph*

Figure 3.2: *Scene graph toy example. Overall modelling scene graph* **(a)** *is recorded as a hierarchy of components that can be instanced and referenced* **(b)**. *Such an organisation is useful for 3D editing where changes on one component are applied to all its duplicates. Note that a tree is just a special type of DAG. Diagram adapted and translated from the GLC Lib [Rib14]. Model courtesy of Marcus Popescu.*

to intermediate between various 3D applications. Likewise, 3D Repo relies on a scene graph as a file-format-independent data representation that can be easily manipulated and stored in a DB. Unfortunately, no universally accepted scene graph definition exists and different implementations impose their own restrictions on the node hierarchy depending on the specific application or rendering requirements. Thus, an abstract interpretation of a scene graph is used in the 3D Repo framework to provide a file-format-independent scene representation as follows. Let scene graph SG be a DAG with a single root node n_{root}, where each node $n \in \text{SG} - \{n_{root}\}$ is stored in its local coordinates with an associated transformation T_n. When recursively applied from n_{root} to each child, these transformations together describe the node's global position in world coordinates. Such a broad definition supports *instancing*, see Figure 3.2, which is required for real-time rendering, i.e. the ability to reduce graphics processing unit (GPU) memory load by *referencing* the same geometry at different locations, but also *automatic merge* functionality which resolves conflicting edits without user's intervention. In this context, a scene graph node represents any domain-specific information such as animations, bones, materials, meshes, shaders, textures, transformations, etc., as well as additional metadata in the form of engineering assemblies, Portable Document Format (PDF) [Ado08] drawings and so forth. To abstract away from this complexity and to futureproof the design, each node is treated as an "opaque" binary document with preserved relational information about the other nodes. Similarly to HyperText Markup Language (HTML) [BFL+14], nodes unknown to the application can be simply skipped to ensure continuous support in the future. Hence, it is possible to add new node types and still be able to retrieve any full or partial graph from the scene regardless of its composition. Unlike 3D files, this polymorphic definition offers the desired flexibility yet establishes a suitable compromise between storage efficiency and querying potential of spatial DBs, see Section 3.3.



(a) *Scene DAG evolution example*



(b) *Revision DAG evolution example*

Figure 3.3: *Scene graph vs. revision history comparison.* **(a)** *Scene DAG evolves over time so that the unmodified nodes (dashed) do not need to be stored again in successive revisions. Those nodes that have been modified (grey) share their SID with common predecessors.* **(b)** *Unlike the scene graph, a revision history DAG explicitly preserves all of its nodes over time. Here, the SID is shared by those nodes that belong to the same branch (grey). Note that each node of the revision history describes an entire scene.*

3.2.2 Revision history

As exemplified by popular file-based VCSs [Sin11], a revision history that supports non-linear branching and merging can, just like a scene graph, be modelled after a DAG, see Figure 3.3. This same data structure enables revisions to have more than one parent while the acyclic property ensures that a revision cannot become its own ancestor thus preventing time travel [Bau08]. However, unlike a polymorphic 3D scene, a revision history graph is composed of only one node type that determines the modelling progress including metadata such as timestamps, commit messages, etc. Therefore, if a heterogeneous scene graph can be stored in a document-oriented DB, so can be a homogeneous revision history.

To achieve this, every graph node regardless of it belonging to a scene graph or a revision history has to specify its unique identifier (UID), a functional requirement of any suitable data store. In a database, the UID would be a unique DB key, while in a file-system, a unique file path. Each node has to further specify its shared identifier (SID) with different meaning depending on the graph it belongs to. Figure 3.3 shows an example. In the context of a scene graph, the SID is shared by all revisions of the same logical scene component. This can be based on the unique name of a component or, in the case of engineering modelling tools, directly on the component's identifier (ID), e.g. globally unique identifier (GUID) in Autodesk Revit [VKR13]. On the other hand, in the context of a revision history, the SID is shared by all members of a single branch. The SID of all zeros, also known as the *null* SID, is reserved for the *trunk/master* to explicitly mark the main development branch.

Together, the unique and shared identifiers form a novel tuple

$$\textit{revision metadata} = (\text{UID}, \text{SID}), \tag{3.1}$$

which is a necessary part of each node in the 3D Repo framework. In relational databases terms, this tuple would constitute a composite primary key on the scene graph and revision history tables so that each scene component can contribute exactly once to exactly one revision and each revision can contribute exactly once to exactly one branch respectively. The possibility of losing such a tuple during the round-trip from the editing software to the 3D versioning system and back is discussed in §3.6.1.

3.2.3 DAG representation

The standard way of representing graphs, directed or not, in computing systems is to express them either as collections of *adjacency lists* or *matrices* depending on their sparsity, i.e. the number of nodes versus the number of connections, as described in the textbook by Cormen et al. [CLRS01]. This has been evaluated alongside some less known approaches in order to establish the most suitable way of representing DAGs in a linear data store such as a NoSQL DB as listed below.

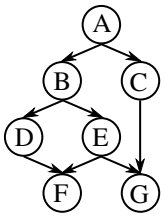
Parental or child links. Storing information about immediate parents or children of a node, also known as the adjacency list, requires recursive hierarchical queries for graph traversal. Unless there is a direct support for this type of retrieval in the data store itself, such an access would have to be implemented at the application level what is computationally expensive due to data transfers.

Adjacency or incidence matrix. Representing graph connectivity as a 2D boolean matrix requires complicated memory management when dealing with large data structures. Given the potentially complex hierarchies of polygonal 3D models and the amount of data each node would have to store, such an approach would not be feasible.

Nested sets. Celko [Cel12] assigns to each node two integers that define the boundaries of all of its children. In contrast to parental or child links, sub-graph retrieval can be implemented using a single query. However, inserting a new entry into the store causes boundary re-indexing on all nodes. This limitation was later resolved by Hazel [Haz08] using real numbers represented as quotients. Nevertheless, these are only suitable for trees where each node has exactly one parent.

Array of ancestors or materialised paths. Storing a full path from the root to each node makes retrieval of sub-graphs relatively easy; all nodes with a given node in their path are its children. What is more, inserting additional nodes requires no updates on the existing entries. Unfortunately, removing nodes requires *re-parenting*, i.e. updating the paths on all of their sub-nodes.

Materialised paths conveniently represent trees in a linear collection of documents. Unlike trees, however, general DAGs can have any number of paths leading from their root to each node which would cause unnecessary duplication when in storage. Hence, to provide an efficient DAG representation and to preserve the advantages of the standard materialised paths notation, an extension is proposed here.



A: [A]
B: [A ▷ B]
C: [A ▷ C]
D: [A ▷ B ▷ D]
E: [A ▷ B ▷ E]
F: [A ▷ B ▷ (D ∨ E) ▷ F]
G: [A ▷ ((B ▷ E) ∨ C) ▷ G]

Figure 3.4: *Extended materialised paths notation example. Each node lists all paths from root* A *to itself. Operator* ▷ *symbolises parent-child relationship. Node* F*, for example, can be reached either via* [A ▷ B ▷ D ▷ F] *or (signified by* ∨*) via* [A ▷ B ▷ E ▷ F].

This reduces the need for duplication by recursively defining multiple-choice paths using `or` notation as illustrated in Figure 3.4.

More formally, any *extended materialised path* is defined by a novel context-free grammar

$$G = \{\{S,A\},\{n,n_{root},\varepsilon\},P,S\}, \tag{3.2}$$

where ε is an empty string, $n \in SG - \{n_{root}\}$ and P is the set of rules

$$S \rightarrow [n_{root} \triangleright A], \tag{3.3}$$

$$A \rightarrow \varepsilon | n | (A) | A \triangleright A | (A \vee A). \tag{3.4}$$

In this new grammar, operator ▷ signifies a parent-child relationship on the graph while ∨ signifies a logical disjunction whenever there is more than one possible path from n_{root} to n. Similarly to standard materialised paths, when reconstructing graph from the selected nodes, the root of a full or a partial graph is the node that has the shortest path as shown in Figure 3.4. This representation is equally applicable to a scene graph as well as a revision history since both can be modelled after a DAG.

3.3 Revision Management

In order to support non-linear histories of 3D models, it is necessary to provide similar functionality to that of a standard file-based VCS, see Sink [Sin11] for examples. Hence, the null SID, introduced in §3.2.2, has been chosen to represent a single trunk/master of the revision history as the main development path. As shown in Figure 3.3, unlimited branches allow for further side developments to be recorded in the same revision history simultaneously. Although both the scene graph and the revision history are represented as DAGs, a crucial difference between the two is that the scene graph makes use of SIDs while the revision history of UIDs for their respective materialised paths. This is because scene nodes share SIDs across time while history nodes share SIDs across branches, as illustrated in Figure 3.3.

Nevertheless, several additional features specific to 3D data have to be considered before a dedicated version control framework can be built. Obviously, there is the need to retrieve the latest `head` revision from every branch as well as the need to identify all previous revisions in a sequence so that any one of them can be queried and recovered independently. With large scenes in mind, the system has to also support partial revision retrieval when only some parts of the original polygonal 3D models are being

queried. This functionality is necessary for the manipulation of scenes where the whole ensemble would not fit into the operating memory or when it is simply not required for that particular editing session. As explained in Section 6.5, it is a common practice amongst modellers to hide certain parts of a large scene while editing. The ability to retrieve only a subsection of a scene is also useful for visualisation purposes when only the closest proximity geometry might be visible at any given time. Next, the framework has to allow for scene graph nodes to be marked as deleted so that they do not contribute to the overall 3D scene from the next revision onwards. Finally, without proper locking, which is not supported in standard repositories such as Git [Cha09] either as it is a file system issue, potential conflicts on scene graph modifications as well as during merging can occur. These are efficiently resolved by 3D Diff described in Chapter 4 that highlights clashing components and offers a fast conflict resolution. However, the framework can easily be extended to enforce access control as discussed in Section 3.6.

3.3.1 Insertion

Initially, when the repository is empty, the entire 3D scene has to be committed as a file-format-independent scene graph data structure, defined in §3.2.1, together with the root node of the history graph carrying essential information about the newly created revision. This information usually consists of the standard commit metadata such as the revision author, message, timestamp, tag, etc., as well as an explicit `current` list of all scene graph nodes' UIDs that belong to the revision. Further commits add new scene nodes and replace their UIDs in the revision which also records SIDs of all those scene components that have just been `added`, `deleted` or `modified`. However, trying to commit changes on any node which is at least one revision behind the head, hence out of synchronisation, results in a conflict. In such a case, the local version has to be *merged* with the head of the repository regardless of the system being centralised like SVN or distributed like Git.

3.3.2 Retrieval

To be able to retrieve any version of a 3D scene from the repository, the `head` or one of its ancestral revisions has to be known. Assuming that the history DAG is represented using the extended materialised paths notation as defined in §3.2.3, the `head` is simply the revision history node that has the longest parental path and has the SID of the desired branch. This is because any ancestral revision of the `head` is listed in the path according to its order of creation. Since each node in the revision history graph explicitly encodes the `current` list of all scene components' UIDs that belong to its revision, once the revision node has been identified, such a list can be queried. Upon retrieval of the 3D components that match the required UIDs, the original scene graph structure can be reconstructed by relying on the same materialised paths notation as before. Similarly, a sub-graph can be retrieved by querying only those scene components that have the desired sub-node's SID in their path.

3.3.3 Deletion

A version control system has to preserve all revisions that have ever been created. Therefore, when deleting a scene component from the repository, it is only marked as `deleted` in the next revision so that it does not contribute to the new `current` list. However, a deleted node is never actually removed

Algorithm 1 Node Deletion

```
function DELETE(node, revision)                          // node to be deleted
    orphans ← FINDORPHANS(node)
    for each orphan ∈ orphans do
        revision.deleted ← orphan.uid
    end for
end function
function FINDORPHANS(node)
    orphans ← node                                       // given node is always orphan
    children ← GETCHILDREN(node)
    for each child ∈ children do
        if ¬ HASANOTHERPARENT(node, child) then
            orphans ← GETORPHANS(child)
        end if
    end for
    return orphans
end function
```

from the storage as it always needs to be accessible as part of those previous revisions which it belongs to. In addition, when marking a node as deleted, its entire sub-graph has to be recursively checked as outlined in Algorithm 1. If any of its descendant nodes have no other parents, i.e. became orphaned due to deletion, they also have to be deleted. Although it would be possible to automatically reparent such orphaned nodes, marking them deleted yields a more robust and predictable system behaviour. Scene graph is a hierarchical structure where node dependencies have direct implications with regards to the resulting polygonal 3D model. Therefore, it is safe to assume that the intention of the user was to delete all remaining sub-nodes unless explicitly instructed otherwise.

3.3.4 Delta compression

Delta compression, also known as *delta encoding* in audio/video formats, is a technique commonly used in VCSs to represent, interchange and store differences rather than whole revisions in order to reduce data size [HVT98, Bau08]. For deltas Δ to be effective, it is important to select their granularity so that the changes are small enough to be meaningful yet computationally tractable, especially on projects comprised of thousands of contributors and millions of edits. Therefore, in the 3D Repo system, the individual scene graph nodes have been chosen to represent delta changes. Even though a single vertex modification would require an entire new mesh document to be stored in the next revision, it can still be considered a delta because only a partial change on the overall scene graph structure has to be preserved. This provides the right balance of complexity and storage convenience.

Since each revision entry records the UIDs of all scene nodes that belong to a particular revision as well as the SIDs for the `added`, `deleted` and `modified` components, access time is reduced to one request to retrieve the revision document itself and another one to retrieve the referred scene components. If the user has a local copy of some previous revision of the 3D scene, only the accumulated `added` and `modified` scene nodes need to be transmitted, akin to a standard *delta combination* which fuses the desired chain of deltas into a single update in modern VCSs [Mac06].

3.3.5 Branching

Since revision history in the 3D Repo system is modelled after a DAG, it also supports concurrent branching. Apart from contributing to the main development path, the users are able to create distinct branches independently of each other. New revisions can be branched off from any previous revision that has been already recorded simply by inserting a new child node into the history graph. This is possible even on a revision that is not the current `head`. Being a new branch is designated by an SID that is different from all other SIDs in the system. Future members of the branch will have to share this new SID which can, depending on the implementation, be uniquely generated even without a centralised coordination.

3.3.6 Merging

Based on the assumption that all scene graph nodes are stored in their local coordinates with associated paths that define their global transformation, see §3.2.1, even changes that seemingly affect the same part of the scene can be merged automatically. Suppose that User_1 checked out the building model in Figure 3.12 and increased its height. By doing so, the roof mesh was displaced but not modified. Further suppose that User_2 added a chimney in the meantime. Due to modifications being independent of their global position, User_2 can commit the changes back to the repository using the auto merge functionality without creating conflicts. However, concurrent edits on the same part of a 3D scene such as modifications of vertices, texture coordinates, materials, etc. by multiple artists are considered conflicting. These cannot be merged without a user intervention. Traditionally, such conflicts are exported into 3D files and resolved manually in a dedicated vertex-level editor. Unfortunately, most popular modelling packages, e.g. Autodesk Maya or Blender, superimpose the models but do not aid the merging process any further. Therefore, a fully automated *3D Diff* tool was developed as described in Chapter 4. Note, however, that in the chimney example, adhering to the building height regulations would be the responsibility of the user merging their changes into the repository.

3.4 Prototype Implementation

The feasibility of the proposed framework is demonstrated on three distinct repository front-ends that support some or all of the revision management and visualisation aspects as described in Section 3.3. The 3D Repo desktop client is a stand-alone application that supports the parsing of common 3D data formats in order to provide a truly file-format-independent 3D version control functionality. On the other hand, the 3D Repo web and mobile clients offer read-only access to the repository in order to render selected revisions on memory limited devices that support OpenGL ES specification [ML10] and its derivative WebGL [Mar11]. The 3D repository itself is built on top of a NoSQL MongoDB [MPH10]. Additional server-side daemon service XML3DRepo is described in Chapter 5.

3.4.1 3D repository

Dedicated graph management systems such as Pregel [MAB+09], Neo4j [RWE13], HyperGraphDB [Ior10] or OrientDB [Tes13] described in Chapter 2 offer large-scale distributed graph processing capabilities. 3D Repo framework, however, does not require complicated graph traversal computations. Instead, it requires an efficient and more importantly a unified way of encoding various object-like 3D

components alongside their revisions in a single centralised repository. Fortunately, the latest document-oriented NoSQL databases provide a suitable alternative to graph DBs in order to store and query scene and revision DAGs as defined in Section 3.2. In contrast to the traditional relational database management systems (RDBMSs), these new DBs preserve structured data with greater flexibility and ease of access while not sacrificing the all-important DB performance and scalability. By design, NoSQL DBs pose no restrictions on the underlying layout of their data collections, i.e. tables in relational DB terms. It is, therefore, possible to store any polymorphic documents in a single collection.

Despite this schema-less approach, it is important to enforce some basic rules regarding the DB structure. Thus, unlike other systems, 3D Repo assigns two collections per repository, one for all the scene graph constituents and one for all those documents that belong to the revision history. Furthermore, the system requires all documents to specify their UID and SID values as defined in §3.2.2.

A standard release open source NoSQL MongoDB [MPH10] has been selected as the data store for the 3D repository. In comparison to alternative JavaScript Object Notation (JSON)-based NoSQL databases such as CouchDB [ALS10] or MarkLogic [Zha09], MongoDB utilises binary documents for storage, querying and data transmission. What is more, this database provides field indexing, full-text search, replica DBs as well as auto-sharding, Map/Reduce functionality and even geospatial indexing making it a suitable addition to existing GIS systems and, therefore, an attractive choice for production environments. Despite all of the aforementioned benefits, MongoDB is not a functional requirement of the 3D Repo framework and could be replaced by any other suitable data store if desired.

3.4.2 BSON encoding

Binary JSON (BSON) [Mon14a] is a little-endian serialised derivative of JSON [ECM13] that employs C data types such as `byte`, `int32/64` and `double` for its terminals. This not only makes the DB highly responsive but also enables additional `date` and `binary` fields which are not part of the standard JSON specification but are necessary for 3D version control. The UID and SID parts of the revision metadata, see §3.2.2, are recorded using universally unique identifier (UUID) as defined by ITU-T [Tel08]. This is especially suitable for version control as it allows committing changes to a single repository without the need for any centralised coordination. Although BSON is designed to be efficient on scanning and parsing, it can occupy more space than a JSON equivalent because it explicitly encodes array indices. Actually, arrays are nothing more than nested documents where each index is an explicit key with an associated array value. Although ordinary 3D attributes such as object name, material shininess, etc., can be encoded directly, the prototype 3D Repo implementation stores long arrays as `binary` entries within BSON documents to avoid unnecessary data overheads.

Hence, 3D vertices V are encoded as an array of triplets

$$V = [(x_1, y_1, z_1), (x_2, y_2, z_2), ..., (x_n, y_n, z_n)], \tag{3.5}$$

where $(x_i, y_i, z_i) \in \mathbb{R}^3$ are coordinates of a single vertex v_i and n is their cardinality. Normals N are represented similarly. However, vertices that correspond to points and lines usually do not define normals

in which case $|V| \neq |N|$. Faces F, on the other hand, define indices into the vertex array V. These are encoded depending on the specified application programming interface (API) level as follows.

If faces define a mix of points, lines and general polygons, then these are represented as

$$F_1 = [k_1, (i_{1,1}, ..., i_{1,k_1}), ..., k_n, (i_{n,1}, ..., i_{n,k_n})], \tag{3.6}$$

where k is the number of subsequent values that are indices i into the vertex array V and together form a face. This encoding corresponds to API level 1. If, however, the faces represent only triangles, this representation can be simplified as an array of triplets

$$F_2 = [(i_{1,1}, i_{1,2}, i_{1,3}), ..., (i_{n,1}, i_{n,2}, i_{n,3})], \tag{3.7}$$

because each face is explicitly defined by 3 indices only. This encoding corresponds to API level 2.

Multi-channel texture mapping UV is represented as one or multiple concatenated arrays

$$UV = [((u_{1,1}, v_{1,1}), ..., (u_{1,n}, v_{1,n})), ..., ((u_{m,1}, v_{m,1}), ..., (u_{m,n}, v_{m,n}))], \tag{3.8}$$

where $u_{i,j}, v_{i,j}$ are texture coordinates corresponding to a vertex j in channel i. There are m channels in total. Similarly, U and UVW channels can be represented.

The byte arrays are assumed to be in a 32-bit little-endian ordering. Listing 3.1 shows an example of a unit cube mesh document using this encoding within a BSON document. Hence, it is easy to perform

```
1  {
2    _id : BinData(3,"1OXvtFihRm6aH+nh+cioGw=="),          // Unique UUID
3    shared_id : BinData(3,"MAU5l5mEVIlCQAAAAAAACA=="),   // Shared UUID
4    paths : [[BinData(3,"MAU5l5mEVIlCQAAAAAAACA==")]],   // All paths from root
5    type : "mesh",
6    api : 1,                                             // API level
7    name : "cube",
8    vertices_count : 36,
9    vertices : BinData(0,"AACAPwAAgL8AAIC/AACAPw... "), // Vertices array
10   faces_count : 12,
11   faces : BinData(0,"AwAAAAAAAAABAAAAAgAAAAAA... "), // Faces array
12   normals : BinData(0,"AAAAAAAAgL8AAAAAAAAAAA... "), // Normals array
13   bounding_box : [[-1.0, -1.0, -1.0],                  // Min xyz
14                   [ 1.0,  1.0,  1.0]]                  // Max xyz
15 }
```

Listing 3.1: *An example of 3D Repo BSON mesh encoding. Please note that the comments (green) are not part of a valid BSON syntax and are only used for explanation purposes. Binary entries for vertices V, faces F and normals N have all been truncated. The full list of all supported scene graph node types with their corresponding entries is available at http://3drepo.org.*

an early reject byte-by-byte memory comparison on the binary fields to determine whether they are identical to those already stored in the repository. If, for example, the number of vertices on a mesh differs from that in the `head` revision, it is flagged as `modified`. But even so, a full binary comparison is still a reasonably fast operation. 3D Repo can, therefore, store in a single database not only the common 3D assets such as meshes and materials, but also textures and even non-standard components such as shaders that would have to be compiled during runtime. However, some of these, e.g. textures, can be extremely large. For instance, the size of a BSON document in MongoDB is currently limited to 16 MB only [MPH10]. This limitation can be overcome using the MongoDB's native *GridFS* functionality to subdivide large documents into 255 KB chunks. Despite the name, GridFS does not correspond to a file system but merely to two data collections inside a single DB. Nevertheless, many 3D formats already impose their own size restrictions. Extensible 3D (X3D) [JIS+13], for example, has a maximum of 2^{16} vertices per mesh. Hence, it is possible to represent components using undivided BSON documents.

3.4.3 Desktop client

3D Repo GUI is an example of a stand-alone application that enables repository management and asynchronous 3D version control. As depicted in Figure 3.5, the client consists of the following software:

1. 3D Repo Core is a portable C++ library that supports low-level data handling and 3D versioning described in Section 3.3. The core provides an abstract DAG definition as well as its two concrete implementations, one for the scene graph and the other for the revision history. Required abstract methods convert these graphs into linear lists of BSON documents that can be stored in a DB.

2. Open Asset Import Library (ASSIMP) [SGK+14] converts more than 40 popular 3D file formats such as Collaborative Design Activity (COLLADA) [BF08], Blender 3D [Bla12], Autodesk 3ds Max [DD13], Wavefront Object [Inc04], etc., into a unified in-memory scene representation. This data structure is in turn transformed into a generic 3D Repo scene graph.

3. GLC Lib [Rib14] supports 3D rendering of hundreds of thousands of components and millions of polygons. This library is used to visualise individual revisions locally even when a third party editing software is unavailable.

4. 3D Repo GUI is the application's front-end written in C++ and a popular cross-platform UI framework Qt [BS08]. The aim of the GUI is to provide a convenient desktop-based 3D version control system without the need to resort to a command line interface.

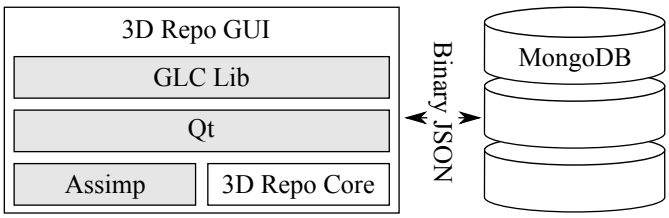
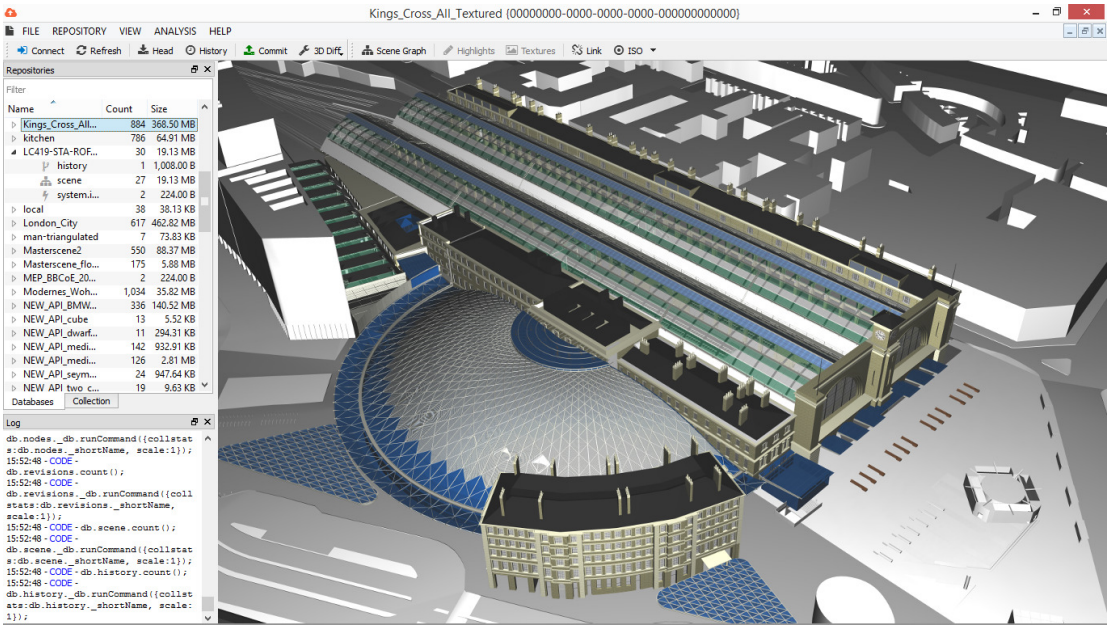
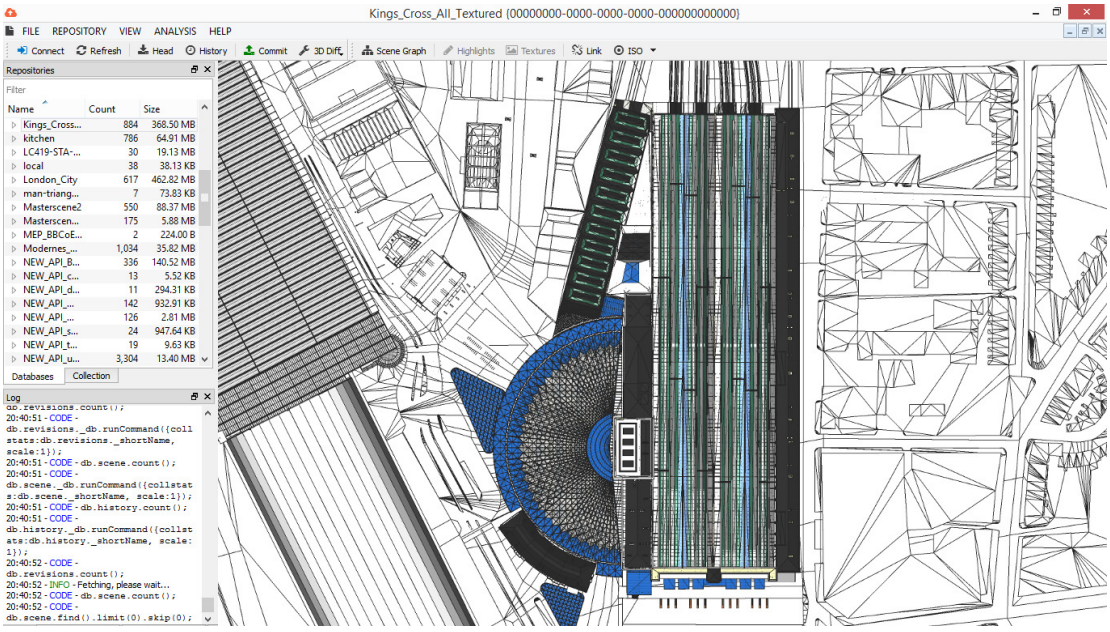


Figure 3.5: *3D Repo GUI client technology diagram. This desktop-based client application utilises cross-platform UI framework Qt and independent C++ libraries Assimp, GLC Lib to manage and visualise 3D revisions from MongoDB.*

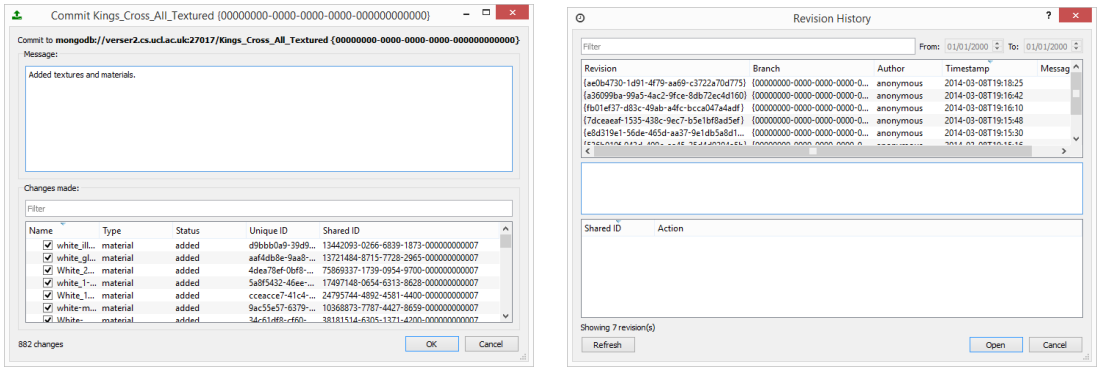


(a) *Head revision*



(b) *Previous revision*

Figure 3.6: *London King's Cross station in 3D Repo GUI. The desktop-based client supports version tracking and asset distribution via a remote 3D repository built on top of a NoSQL database MongoDB.* **(a)** *The* `head` *revision loaded from the* `master` *branch.* **(b)** *Previous revision in orthographic projection. Model courtesy of Network Rail.*



(a) *Commit dialog* **(b)** *Revision History dialog*

Figure 3.7: *3D Repo GUI dialogs.* **(a)** *Commit dialog listing all scene graph components that are to be committed in the next revision.* **(b)** *Revision History dialog listing the revisions and associated metadata.*

By utilising these components, the GUI, shown in Figure 3.6, is able to act as a general 3D model viewer and file-format converter while supporting scene graph delta commits, revision retrieval and automatic conflict resolution as defined in Section 3.3. On load, the user connects to a remote repository using the host and port values and user-specific credentials which are required on a per-project basis or globally if authenticated on the admin database. In a single session, the user can be connected to multiple hosts for easier cross-domain data manipulation. The contents of the repositories, i.e. individual version-controlled projects, are listed in a repositories dock panel together with the count of nodes and the disc space occupied, see left-hand side of Figure 3.6a. These can be filtered using a built-in full-text search. Underneath, there is a current log output preserved locally for auditing purposes. Multiple revisions can be opened and manipulated simultaneously via interlinked navigation, see Figure 3.11. On commit, the user uploads only the most recent delta changes via automatic conflict resolution as defined in Section 3.3 resulting in the commit dialog shown in Figure 3.7a. However, the GUI does not support any vertex-level editing so if a more complicated merge is required, the conflicting revisions have to be exported into a separate 3D file for external editing. Once in the repository, several revisions can be retrieved simultaneously using the revision history dialog as shown in Figure 3.7b. The GUI is fully multi-threaded and each action is performed per open window without freezing the whole user interface.

3.4.4 Web client

3D Repo web client is a proof of concept example of a lightweight read-only repository front-end that visualises revisions in web browsers. The purpose of this client is to connect directly to the database in order to provide a simple and efficient way of rendering selected revisions without the need for a stand-alone application server.

Even though plenty of work has been done on the server-side management of MongoDB, there currently exists no standardised JavaScript library that would communicate with the DB from the client-side [MPH10]. Since BSON specification is not yet supported by modern web browsers, one option is to rely on the MongoDB Node.js [Ihr13] parser [Mon14b]. Another approach, however, is to utilise the native MongoDB Java driver which not only decodes the BSON data format but also handles the Transmis-

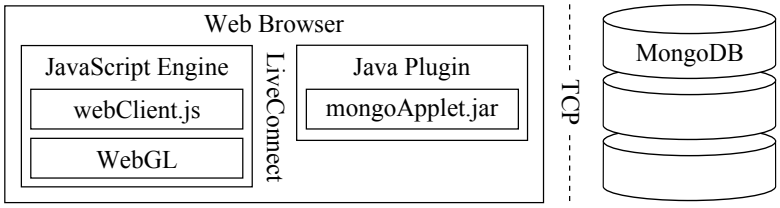


Figure 3.8: *3D Repo web client technology diagram. Client-side JavaScript leverages the MongoDB Java driver (exposed via a custom applet) to query BSON documents from a 3D Repo. JavaScript-to-Java communication is possible thanks to the LiveConnect feature of the Java browser plug-in. Once the scene graph is reconstructed, WebGL renders the retrieved geometry inside a web browser.*

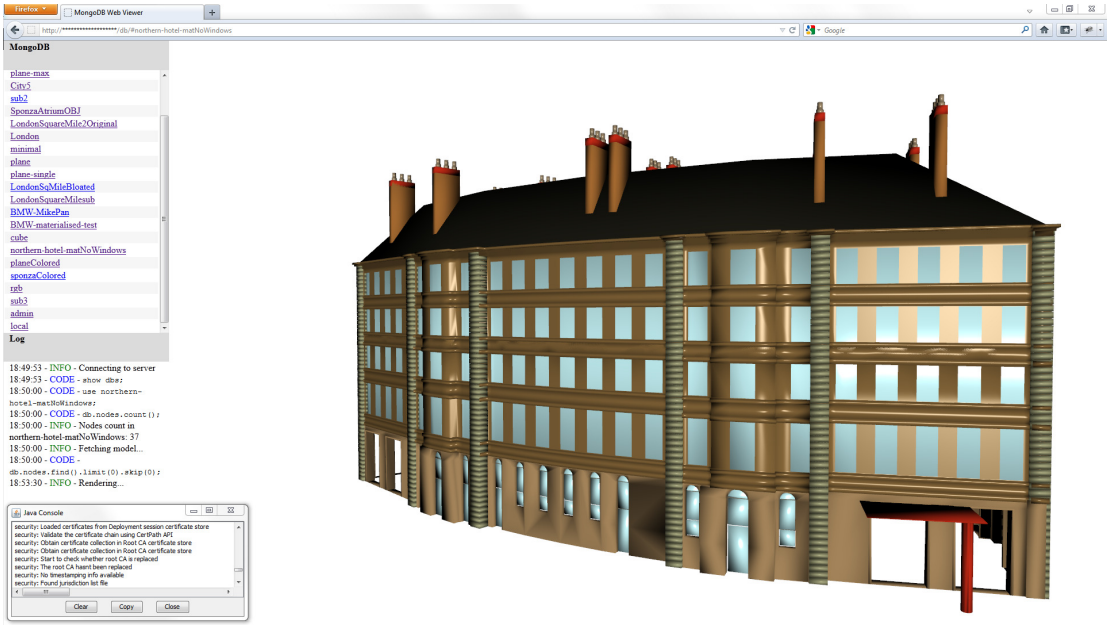


Figure 3.9: *3D Repo web client rendering the Great Northern Hotel 3D model. This lightweight client visualises the contents of the repository in WebGL-enabled web browsers without the need for a dedicated application server. Note that this proof of concept application has been superseded by a more advanced XML3DRepo client discussed in Chapter 5. Model courtesy of Network Rail.*

sion Control Protocol (TCP) DB connections. This is possible thanks to the *LiveConnect* [Ora12a] Java plug-in feature that enables communication between the Java Virtual Machine (JVM) and the JavaScript engine running in a web browser. On one hand, a Java applet can invoke scripts in a HTML page and populate JavaScript objects. On the other hand, JavaScript can access Java runtime libraries, static methods, create objects and execute public methods on Java applets. Therefore, the web client is written as a combination of a default MongoDB Java driver, JavaScript and WebGL as depicted in Figure 3.8. The driver is wrapped into an applet that exposes useful methods to JavaScript but has no execution logic to manage the client itself. Additional static methods convert between the BSON document collection retrieved from the repository and a scene graph representation required by WebGL. Once the scene graph is retrieved and reconstructed, the JavaScript application renders the polygonal 3D model using WebGL as depicted in Figure 3.9. However, as a safety measure, LiveConnect restricts the ability of JavaScript to open any cross-domain Java connections, see the security model reference in [Ora12a]. Thus, the 3D Repo web client has to be loaded from the same domain as the DB, a small yet important limitation.

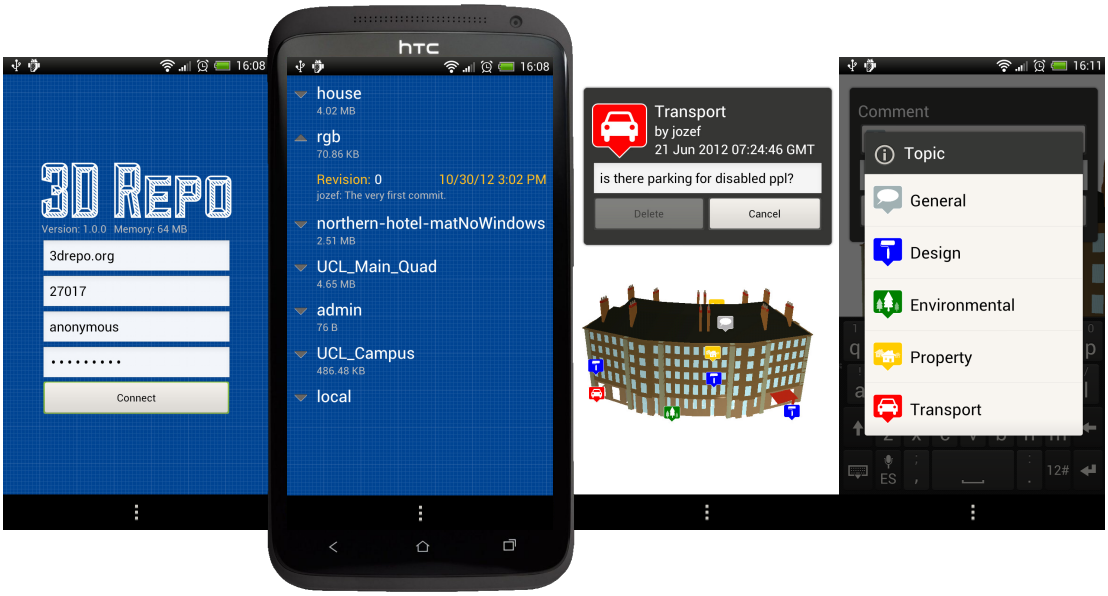


Figure 3.10: *3D Repo Android app available in Google Play Market. The user logs in to a remote 3D Repo data store, selects a specific revision, previews the existing comments and commits new comments back to the repository. This application was developed for the purposes of public enquiry during which the members of the general public can voice their concerns in regards to public development. Visualised here is the Great Northern Hotel which is a part of the King's Cross station redevelopment project. Model courtesy of Network Rail.*

3.4.5 Mobile client

Although most modern mobile devices support the respective technological components of the 3D Repo web client, LiveConnect bridge between Java and JavaScript is currently not available beyond desktop web browsers. Therefore, an Android-specific [GN14] 3D Repo mobile client visualises centrally stored polygonal 3D models on mobile devices.

Since Android is inherently Java-based, the MongoDB Java driver from §3.4.4 was ported to the Android platform. A custom rendering engine based on OpenGL ES [ML10] was developed to query the scene and update user annotations via an interactive 3D preview. In order to offer a scalable visualisation platform, decomposed assets are streamed onto client devices, reconstructed into a suitable 3D representation and displayed for viewing.

Similarly to the desktop 3D Repo client, the user first selects the host and port to connect to and provides user-specific credentials to log into the repository. Then, the user selects a project to visualise. Once the scene is loaded, they can freely navigate the 3D space and read and write comments at any location. As shown in Figure 3.10, these are divided into five main categories identified by distinct icons. Localised comments are pushed back to the repository for subsequent analysis. An overview of the scene shows hotspots where most of the comments were posted. This concept is expected to improve the public enquiry process and significantly reduce the costs of running such events, see Chapter 1.

Figure 3.11: *72 revisions of the UCL Cruciform building version controlled in 3D Repo. Random colours have been applied to highlight manifold surfaces and their corresponding features. Model courtesy of Jules Bodenstein.*

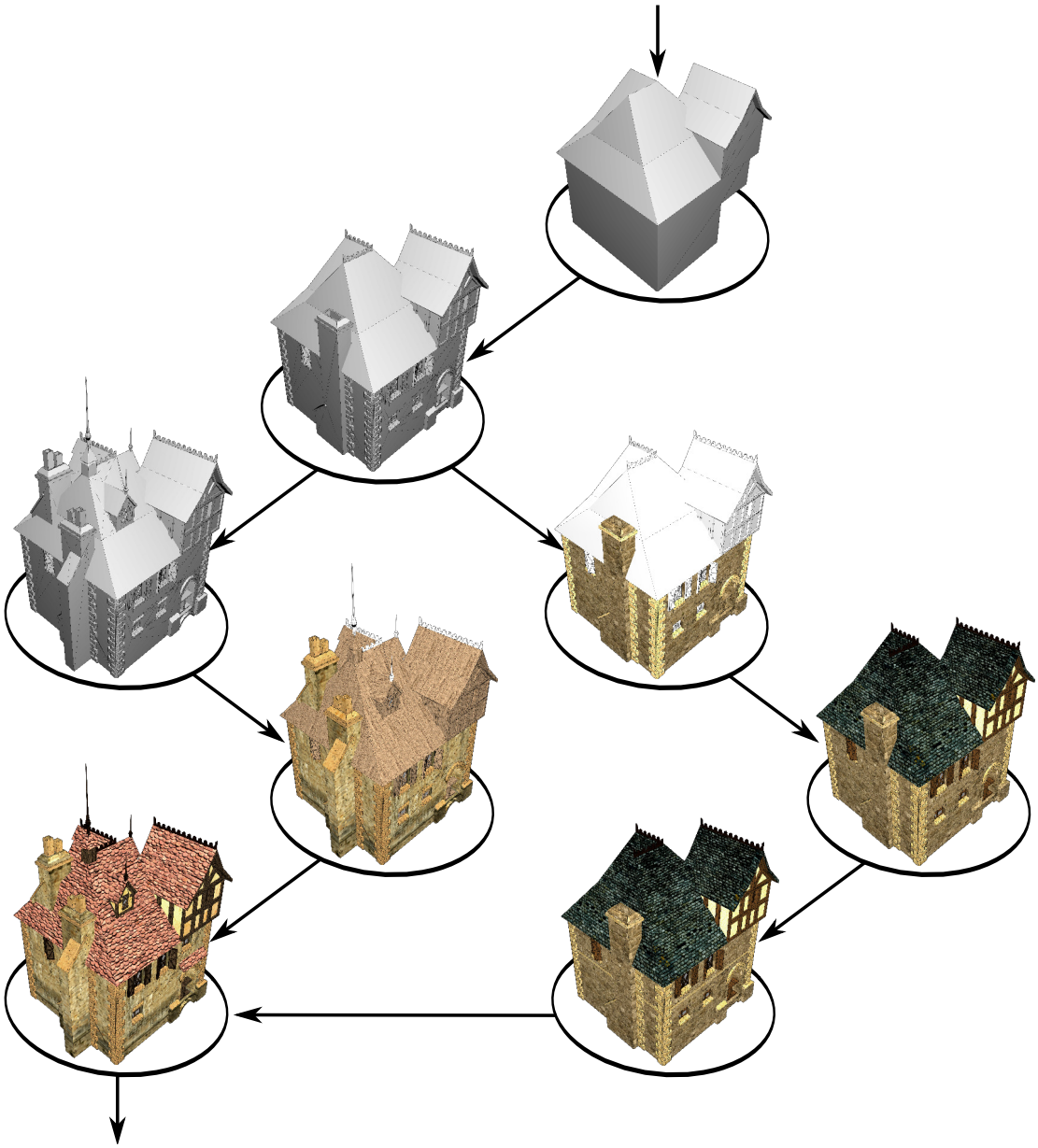


Figure 3.12: *Revision history snapshot recorded using 3D Repo. Model courtesy of Johnathan Good.*

3.5 Evaluation

Evaluated here are some of the main features supported by the 3D Repo framework and the associated client applications presented in Section 3.4.

- Visualised within the 3D Repo GUI in Figure 3.6 is an example of a large architectural 3D model with 3,171,115 polygons and 2,439 meshes that is version controlled in a remote 3D repository. The original scene created in Autodesk 3ds Max [DD13] occupies over 3.5 GB of disk space and is spread across 595 files, 365 of which are textures. In contrast, 3D Repo preserves the same scene in MongoDB using mere 368.5 MB of storage. All assets including meshes, materials, transformations and textures are encoded as BSON and are fully version controlled. Hence the requirements for storage are significantly smaller than other versioning methods described in Chapter 2.

- As shown in Figure 3.9, it is also possible to retrieve a subset of a large 3D scene from the repository and visualise it within a web browser. This model, the Great Northern Hotel, is tracked through time independently of the main scene, the King's Cross station. Unlike the desktop client, the web browser viewer provides a read-only access to the repository. Although this proof of concept prototype demonstrates the feasibility of rendering the contents of the repository in web browsers, it has been superseded by XML3DRepo and its associated viewer, see Chapter 5.

- Figure 3.10 demonstrates the collaborative nature and the scale at which the 3D Repo framework can reach potential users. Using the mobile client, hundreds of stakeholders are able to seamlessly collaborate on a single project by placing virtual comments directly into the 3D scene.

- Figure 3.11 demonstrates a large number of revisions managed by 3D Repo. The desktop-based client is able to visualise all revisions simultaneously with interlinked user navigation. Each of the revisions can be exported as a stand-alone 3D file for subsequent local editing.

- Finally, Figure 3.12 demonstrates an example of a non-linear history being tracked via 3D Repo. Features from a side branch are automatically merged into the main development stream of the `master` branch. The Medieval dataset was created in Trimble Sketchup [Sch13] and imported to 3D Repo using COLLADA [BF08] file format.

3.6 Discussion

Interaction between the modelling software and a domain-specific 3D version control framework 3D Repo takes place through the import and export of files via the main desktop-based application 3D Repo GUI and other web-enabled lightweight clients. The software architecture and prototype implementations presented in Sections 3.2 and 3.4 respectively succeeded in decoupling the modelling from its long-term storage, as files are now considered to be only temporary representations of information in order to facilitate data interchange with a variety of editors. This is of great importance as the range of editing software and other mesh generation tools is already vast and likely to grow even more in the future. Rather than relying on vendor-specific features such as revision histories stored in the native Autodesk Maya files or traditional file-based VCSs that are inherently not suited to the diversity of 3D assets, the users of 3D Repo are able to track revisions and store only the deltas rather than whole files.

3.6.1 Limitations

By definition, the smallest unit of change in the 3D Repo system is a document. In a mesh, binary arrays are used for vertex coordinates, normals and face indices, see subsection 3.4.2. This representation is efficient if the likely access point is a collection of documents, but is not once the edits become very localised. If, for example, a single vertex has been repositioned, the whole document but not the entire 3D scene would have to be replaced in the next revision. This level of granularity might not be suitable to all projects, although it would certainly be possible to support multiple types of object representations within the same repository. The UI could store small edits as operations depending on the structure of the DB collections (tables). In addition, when interacting with the framework, existing

file representations need to convey changes efficiently. That is, the system must be able to detect that the imported scene components match the existing entries in the repository and that some of them have changed since the previous revision. At the moment, the system relies on the assumption that imported assets preserve revision metadata UID and SID for each component. However, there is no guarantee that the editing software will do this and users will always be able to modify the data if they wish. Thus, an interesting matching problem occurs. It is possible to compare scene graph nodes against all their previous revisions stored in the DB. This can be speeded up by relying on the data sizes as an early indicator of correspondence and the fact that matching is a pure binary comparison. To go even further, it would be necessary to support the matching of arbitrary meshes against each other, an interesting research problem in itself which can be easily circumvented by a direct editor connection, see §3.6.2.

The main limitation of the current proof of concept applications presented in Section 3.4 is the recurring direct DB connection. Since no functional changes have been made to the DB itself, the desired version control logic had to be placed at the application level. Unfortunately, schema-less NoSQL databases like MongoDB lack data validation on insertion, unless specified as a unique key. Placing the application authority client-side gives rise to potentially inadvertent damage on the repository as each client gains a raw access to the data store. This limitation is addressed in Chapter 5 through the development of a server-side daemon and a corresponding Representational State Transfer (REST) API.

3.6.2 Extensions

As suggested in Figure 3.1, GUI applications are not necessary to manage a remote 3D Repository. The applications that connect to the DB and facilitate revision control are mainly prototype examples that demonstrate the feasibility of the proposed solution. The editing software could itself replace the desktop GUI. Plug-in frameworks provided by many modern editors such as Autodesk 3ds Max or Blender could connect to the DB directly, exploiting all the data available in the repository. In particular, on committing changes back to the DB, the system would know exactly at which point an asset was retrieved and which revision it belongs to. Thus, the internal workings of the revision control could be entirely hidden from the user. It is important to note that unless some form of access control is in place, conflict resolution is still a required part of such a framework. An obvious feature that could be added to the system is access control via locking. Even though it is not difficult to add locks to the existing DB schema, it is difficult to preserve them in a round-trip to the file store. What is more, it is not trivial to decide on the granularity level of the locks themselves. One can imagine supporting read-write locks that would prevent conflicts arising altogether. Hybrid types of locks, on the other hand, could enforce geometric constraints on the 3D model. A single polyline in such a case could define a locked interface between a piece of geometry to be modified and the rest of the scene. Addressing this is planned for future work, see Chapter 7. Another feature to add is the facility to search for assets via spatial queries, reminiscent of the role of spatial DBs. To achieve this, a bounding box hierarchy could be provided. This could be used for both general queries such as “fetch all objects within this region”, “fetch all objects adjoining given object”, but also to facilitate the detection of matches when revision metadata has been lost. MongoDB already offers native geospatial longitude and latitude indexing to further support it.

3.7 Chapter Summary

The first research question in Chapter 1 asked whether collaborative 3D editing can be scaled to useful model sizes. Thus, the framework presented in this chapter explored a novel approach to storage and asynchronous revision control of 3D assets using a document-oriented NoSQL DB. Three different DB front-ends have been developed in order to enable non-linear version control specifically designed for 3D assets, automatic merging as well as remote repository visualisations. It is believed that the unification of storage and versioning in a single DB can provide significant benefits over the common practice today which uses traditional VCSs for asset tracking. 3D Repo system, therefore, decoupled polygonal 3D modelling from its long-term storage by creating a suitable repository in a standard release MongoDB. Firstly, the issues of representing DAGs in a document-oriented DB was resolved. Next, the size and types of elements stored were selected. On top of this, a revision control system was created that allows for branching and merging and enables concurrent 3D modelling over the Internet. When creating the framework, support for editing using the broadest range of tools was decided; thus relying on import and export of files rather than relying on specific modelling tools. In addition, the revision history was captured alongside individual 3D assets within the same repository.

The main contribution of this chapter spans from identifying and exploiting similarities between a scene graph and a revision history. Since both data structures can be represented as a DAG, it is not only possible to store large 3D scenes in a remote repository but also preserve their editing history at the level of individual components. Feasibility of the proposed solution was demonstrated on numerous examples with up to several millions polygons. The system is able to load over 40 different file formats and store them in a unified scene graph representation in a NoSQL DB. In addition, each asset is version controlled using unique identifiers (IDs) so that multiple users can collaborate on the same scene concurrently.

It is, therefore, fair to argue that this approach would be a suitable way of establishing large-scale collaborative development and visualisation as asked in research Questions 1 and 2 in Chapter 1. The prototype tools presented in Section 3.4 are already useful for management and integration of a few million polygon models, in that multiple versions can be stored in one place, and different users can access the models remotely. Given the type of database in use is proven to scale massively, the potential long-term benefits include the ability to serve a large number of editors using a single scene, or even open crowd-sourcing of 3D models, see Chapter 5.

However, the 3D Repo system does not yet support per asset locking. Hence, a form of visual 3D differencing and merging interface is a crucial component of the non-linear 3D version control. The 3D Diff UI, described in Chapter 4, is able to utilise the knowledge about a common ancestor of two 3D models in order to effect a 3-way merge which automatically resolves conflicting edits that would otherwise be ambiguous. It further introduces the notion of *implicit* and *explicit* conflicts in order to preserve semantic relationships in 3D models. Such a functionality is crucial for easy maintenance of 3D repositories where hundreds of users could collaborate on the same 3D scene simultaneously. Nevertheless, such tools including client applications presented in Section 3.4 require a direct DB connection. Chapter 5, therefore, introduces a daemon service REST API which acts as a gateway to the system

and a layer of indirection on the server-side. Finally, to extract provenance from legacy datasets, a 3D Timeline interface, presented in Chapter 6, reverse engineers editing histories from stand-alone 3D files. Since this algorithm makes no assumptions about the scene structures, it is able to compare different file formats simply based on their geometry.

Chapter 4

Visual 3D Differencing and Merging

Apart from application of a domain specific VCS to management of 3D assets, the first research question in Chapter 1 also asks whether it is possible to sustain collaborative editing without the need for per-asset locking. As modelling software grows in use, and as 3D models get more complex with input from many users over time, there is an emerging problem of maintaining scenes that were edited concurrently. Thus, in order to support non-linear history of polygonal 3D models, an interface for visual 3D differencing and merging is required. Although 3D Repo framework, introduced in Chapter 3, is able to effectively preserve non-linear revisions and branches alongside the main development path, it still depends on a convenient and user-friendly tool to identify and manage concurrent edits. A 3D scene might be edited by different users simultaneously, hence, there is the need to merge several revisions to create a consistent output preserving the desired changes from each user. Yet, most of the time it is possible to simply take whole sections from respective models to form a coherent result. Such an approach is convenient, especially, when manipulating complex scenes. It is also less prone to merging errors as it manipulates fixed scene components rather than low-level vertices. In general, comparing polygonal 3D models is a complex and time consuming task. When combining modified versions of the same scene, popular 3D modelling packages show the models, enable their editing but do not assist the merging process in any way. This chapter, therefore, presents a novel tool, *3D Diff*, that supports visual differencing and merging of 3D assets. Here, the problem is framed in a way that is analogous to source code conflict resolution. Firstly, as described in Section 4.2, the tool automatically detects the differences in polygonal 3D models by noting the correspondences and discrepancies between them. Secondly, it provides an interactive UI to select between changes in order to effect a merge. To achieve this, the novel notions of *explicit* and *implicit* conflicts in polygonal 3D models are introduced in Section 4.2, and a prototype implementation that supports a novel conflict resolution process is developed, see Section 4.4. This performs a bounding box clash detection and 3D visualisation and allows its users to quickly select one of the revisions from each of the conflicted scene graph nodes. By further integrating the knowledge about the immediate common ancestor of the differenced models, also known as a *3-way diff*, the tool is able to automatically resolve more conflicts than in a standard *2-way* comparison. This is evaluated in a pilot user study in §4.5.1, the results of which suggest that 3D Diff is an effective way to merge polygonal 3D models. Section 4.6 thus concludes that such tools have an important role in the maintenance of large 3D scenes.

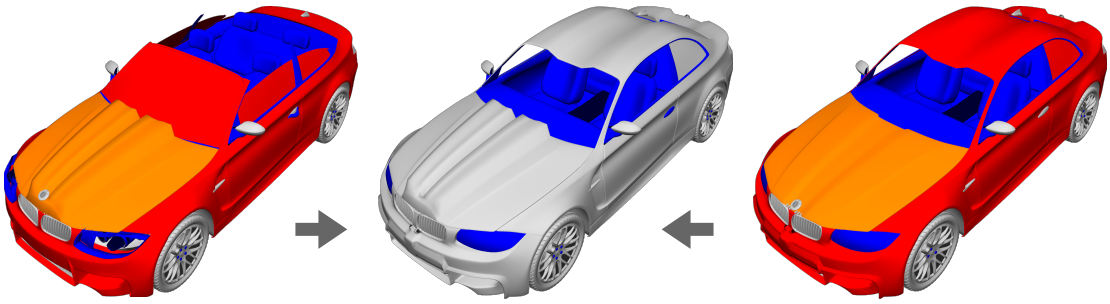


Figure 4.1: *3D differencing and merging via 3D Diff. Two revisions, left and right, of the same 3D model are compared. Conflicting edits are highlighted in red, non-conflicting modifications in blue, user selection in orange and manually merged results in grey. Model courtesy of Blender Foundation.*

4.1 System Overview

Computers have become more capable in the scales and richness of graphics that they support. The complexity and size of 3D models as well as the amount of individuals collaborating on the same scene simultaneously have grown dramatically over the past few decades. This explosion in content creation is easily witnessed by a large number of models available in repositories such as Sketchup 3D Warehouse [Tri14] or TurboSquid [Tur14]. Thus, the emerging problem is that of maintaining a set of 3D models over multiple artists working on the same repository concurrently. Existing popular modelling packages such as Autodesk 3ds Max [DD13], Maya [Pal13] or Blender [Bla12], however, do not natively aid this process and provide only manual means of conflict resolution. The most common approach is to superimpose the models and delete scene components one by one until the desired merge is achieved, see Figure 3.12. Unfortunately, without an automated conflict detection, such an approach would not be feasible for large scenes where there are often thousands of components present, see Figures 3.6 and 5.4.

The problem of concurrent 3D editing can be compared to an analogous situation in software engineering. When multiple users edit the same file concurrently, one has to perform automated differencing in order to identify the conflicting changes. Many *delta algorithms*, i.e. procedures that detect changes for the purposes of version control and compression, have been proposed over the years [HVT98, BMZ+05]. Discrepancies identified using these algorithms are then resolved either by accepting or rejecting *"theirs"* or *"mine"* whole revisions on a per file basis or by modifying conflicting files line by line.

By way of comparison, in the domain of computer graphics, the novel concepts of *3D differencing and merging*, depicted in Figure 4.1, describe the process of identifying and resolving changes between two or more input 3D models that might be comprised of multiple binary or text files. Commonly, these files tend to describe the geometry, material properties, textures, etc., and even application specific data such as shaders. Traditional VCSs such as SVN or Git can store all such files, however, their merging tools are not designed to support diverse types of 3D assets. Whilst for small polygonal 3D models it might be possible to perform a line-based merge as described in Chapter 2, the meaning of changes in large scenes is almost impossible to comprehend and visualise using existing tools and processes. Furthermore, it is not always possible to construct a consistent scene by only selecting parts of the input models because 3D edits might interfere with each other. This is of similar character to

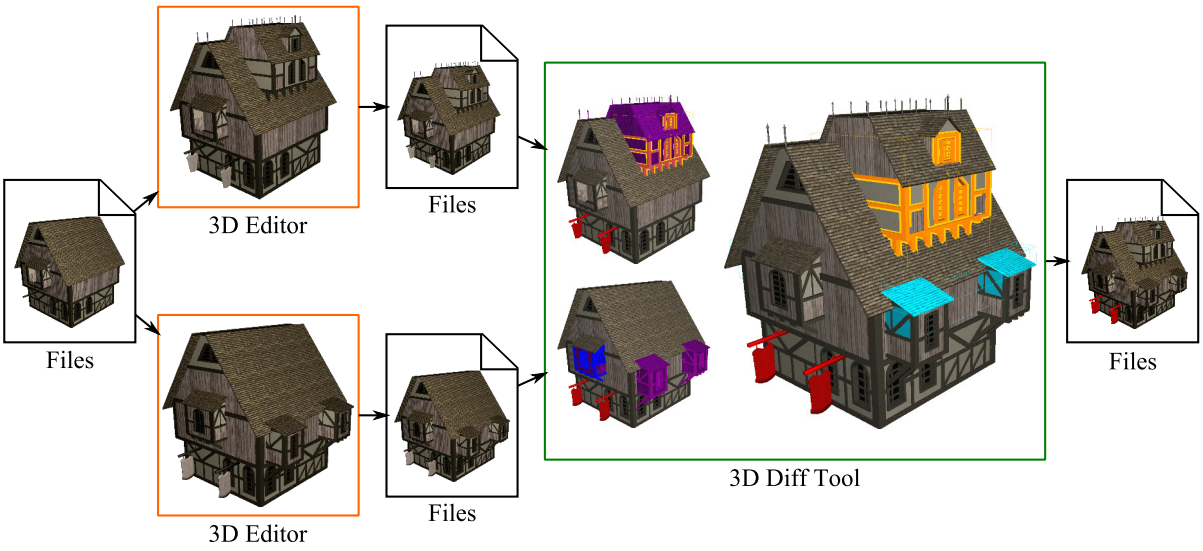


Figure 4.2: *3D Diff processing pipeline. 3D models are edited concurrently in standard modelling packages such as Blender or Autodesk 3ds Max and exported as files. These are loaded into a stand-alone 3D Diff tool which performs differencing and interactively resolves conflicts into a merged result. The result can then be exported as a new file or simply uploaded to a 3D Repo versioning system.*

software merging where a naïve approach may not preserve the semantics and structure of the source code [JL94, LHKR12]. A prototype 3D Diff tool presented in Section 4.4, therefore, identifies *explicit* conflicts whenever the corresponding scene components have been edited differently, as well as *implicit* conflicts that are caused as side-effects of the merging process itself. Although the concept of implicit conflicts has no direct relationship to the source code, it is related to the detection of the aforementioned semantic conflicts. 2-way and 3-way 3D differencing and merging are performed similarly to their software engineering counterparts [Men02].

Section 4.2, thus, describes how two polygonal 3D models and, optionally, their common ancestor are compared by their scene graph representations down to individual discrepancies at the scene node level. Preserving the syntax and semantics of the models using such an approach is relatively straightforward as standard scene graph conventions isolate the geometry and associated metadata into units that can be shared across different systems and applications, see Section 3.2. By respecting the version control actions of *adding*, *deleting* and *modifying* components, the semantics of a scene graph are being preserved. However, structural changes made to a 3D scene, e.g. re-parenting of an object, can potentially make no visual but huge structural changes to the model files.

Fortunately, unlike standard VCSs that deal with files, 3D Repo, described in Chapter 3, is based on a database of scene graph components and treats files only as temporary data representations. This ensures that structural changes have no impact on the storage requirements of the repository or the version control logic. Nevertheless, the output of this kind of high-level 3D merging may not always be completely resolvable, hence, in rare cases where vertex-level editing is required, this is expected to take place in an external 3D editing tool.

4.1.1 Processing pipeline

As depicted in Figure 4.2, the input to the processing pipeline are two polygonal 3D models that are to be differenced with an optional addition of a third model, their common ancestor. Although the ancestor is neither necessary, nor is it displayed to the end user, in a 3-way diff, it helps the system to resolve the ambiguous cases where there are scene nodes missing in one of the input models. These cases are ambiguous because just from the two revisions it is not always possible to reliably determine whether a new object has been added or whether an old object has been deleted. Even though 3D models, especially when merging branches, tend to come directly from a remote 3D repository instead of a local file system, somewhere along the revision history the data must have been processed as files using modelling packages, otherwise no changes would have been recorded. Therefore, the prototype 3D Diff interface, described in Section 4.4, supports polygon meshes with materials and textures loaded either from local 3D files or from a 3D Repo repository through a file-format-independent scene graph representation introduced in §3.2.1. Extensions to other forms of 3D asset types are a matter of implementation only and does not present any specific challenges other than parsing and visualisation. At the end of the pipeline, the output is a new polygonal 3D model with some or all of the conflicts resolved by taking entire changes from one of the two conflicting inputs akin to standard merging in source code versioning. Hence, the process that leads to syntactically and semantically correct combination of multiple edits across several revisions of the same 3D scene consists of two consecutive stages as follows:

1. The initial stage, described in Section 4.2, automatically detects discrepancies between the given models and highlights them visually as a colour-coded overlay on top of individual scene components in a 3D Diff GUI. In its default mode, the algorithm performs 2-way differencing comparing only two polygonal 3D models. The inclusion of the ancestor of the differenced models further informs the process and helps to resolve some of otherwise unresolvable edits, see Section 4.2.

2. Once all changes have been determined, automatic merge suggestions are proposed to the end user, see Section 4.3. In cases where direct conflicts on the same scene graph node exist, the user has to either accept one or the other revision, or leave both conflicted components within the output model and resolve them manually in an external 3D editor.

4.1.2 Scene node correspondence

Large 3D scenes tend to encompass multiple separate components rather than a single manifold surface, see Section 3.2 for a discussion. 3D Diff leverages this natural partitioning and compares scenes via their corresponding scene components. Hence, various 3D file formats are converted into a unified scene graph representation such that each node represents a delta change. If, for instance, a single vertex has been repositioned in a mesh, this is considered a modification and the entire mesh has to be highlighted for user's attention. In order to provide an automated detection of changes across two revisions of the same scene, it is necessary to establish an individual node-to-node correspondence. That is, for each node in one graph, 3D Diff has to find a matching node in the other. Some nodes will be the same, some will exist in one graph but not the other, whilst some will occupy the same place in the scene hierarchy

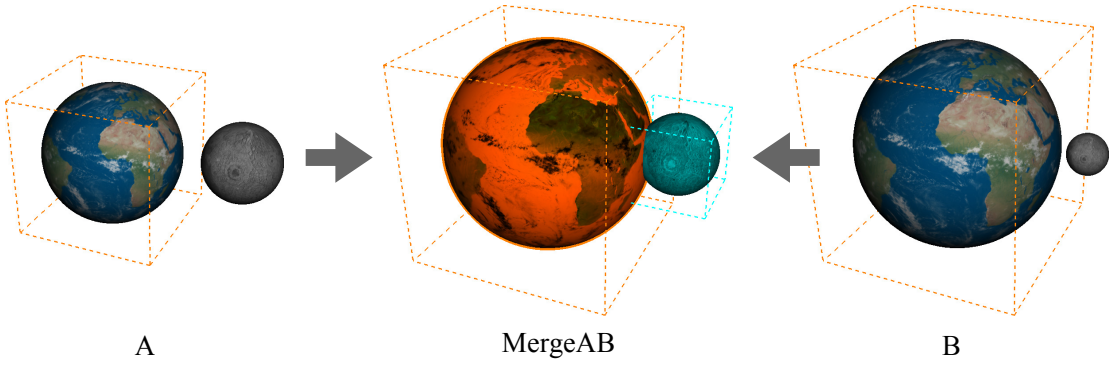


Figure 4.3: *An example of an implicit 3D conflict. Here, two users expanded the Moon (left) and Earth (right) independently of each other. When merging, 3D Diff highlights bounding box intersections that did not exist in either of the differenced models but were introduced during the merging process. Orange shows the current user selection, while turquoise the offending change.*

but a different region in a 3D space. Since 3D Repo system assumes SID revision metadata, defined in §3.2.2, assigned to each scene component, it is possible to establish a precise correspondence across two 3D models. For an alternative approach that makes no assumptions about the scene structure or metadata especially when dealing with legacy datasets, see Chapter 6. Nevertheless, one of the main features of a scene graph is its ability to instance components.Thus, if a node has been modified, all of its instances get affected equally. If this was not desirable, the modeller would have split the instances beforehand.

4.2 3D Differencing

In order to determine what exactly has changed in different revisions of the same polygonal 3D model, 3D Diff compares the corresponding scene graph components in a pair-wise manner. This automated process is analogous to the line-based *Diff* algorithm by Hunt et al. [HM76], but is designed specifically for 3D models rather than the source code. Unlike the original algorithm, however, 3D Diff starts with two scene graphs comprised of different node types, where the correspondence between them is known based on their shared identifier (SID) values as defined in §3.2.2. Instead of lines of text, the conflicts defined in this section are detected on parts of 3D models regardless of their file format representation.

Explicit conflict. An explicit conflict in a part of the same 3D model in two of its revisions means that the part exists in both models but is not identical. Part equality and its granularity is implementation dependent. In the 3D Repo framework, for example, the lowest level of change is a scene graph node. Although it would be certainly possible to redefine the system to deal with individual vertices rather than whole components, it may not be suitable for a dedicated 3D version control framework. This is because changes on vertices on their own do not necessarily express any high-level 3D editing operations. What is more, assigning a per vertex correspondence across two modified 3D models is a non-trivial task, too. Performing subdivision, for instance, could replace all vertices meaning no direct matches would exist. See comparison with [DP13] in §6.5.2. However, most engineering packages work with geometry at the object level, so vertex-level differencing is not an explicit user requirement in Section 1.2.

Implicit conflict. An implicit conflict, on the other hand, means that the semantics of a 3D model are somehow violated. Imagine that two users expanded the Earth and Moon meshes in Figure 4.3 inde-

pendently of each other. Even after these expansions, the meshes did not overlap in either of their new revisions A and B. However, it could well happen that once such changes are put together in a new combined revision *MergeAB*, they would intersect. Whilst it is not always possible to infer all such implied violations, a reasonable approach is to observe the changes in collisions between 3D objects. If two objects collide after a merge, where they did not before, they are likely candidates for an implicit conflict. It is reasonable to expect that any intersections in an input model are intentional as otherwise the modeller would have not left them there. However, when merging changes from different revisions, new intersections might be introduced that did not exist before. Since the bounding box clash detection is a fast and common test in 3D graphics, it can be easily used for indirect conflict check on any geometry.

Hence, suppose that different meshes n and k both present in two distinct 3D model revisions A and B have been independently modified but did not have a bounding box intersection to begin with, as shown in Figure 4.3. If in their merged result *MergeAB* they do have a bounding box intersection, this is considered an implicit conflict. Therefore, an implicit conflict occurs whenever the following holds:

$$\square(n_A) \cap \square(k_A) = \emptyset, \tag{4.1}$$

$$\square(n_B) \cap \square(k_B) = \emptyset, \tag{4.2}$$

$$\square(n_{MergeAB}) \cap \square(k_{MergeAB}) \neq \emptyset, \tag{4.3}$$

where operator $\square(m_R)$ determines a bounding box of a component m in a model revision R and $n_R \neq k_R$. By counterexample, if $n_{MergeAB} = n_A$ intersects $k_{MergeAB}$, but $k_{MergeAB} = k_B$, then $k_B = k_A$ would be ignored because it was an intentional change in A. The same holds vice versa for n_B and k_A by symmetry. Thus, instead of checking all possible bounding box intersections in all the meshes of A, B and *MergeAB*, it is sufficient to only compare those meshes that have been modified leading to the same result with a significantly smaller amount of pair-wise checks. As this method does not rely on the common ancestor of the input 3D models, it is equally applicable to 2-way and 3-way 3D differencing. Nevertheless, it is not applicable to materials or textures as they have no equivalent of a 3D intersection.

4.2.1 2-way diff

A standard 2-way 3D Diff compares two polygonal 3D models, highlights the differences and ignores commonalities across scene components. Let A and B represent two 3D revisions, i.e. respective sets of scene graph nodes in two distinct 3D models, and n a single corresponding node which can change across these revisions. Hence, for a 2-way diff, the only possible states of a component n are as follows:

$$\text{unmodified} \Leftrightarrow (n_A = n_B \wedge n_A \neq \emptyset \wedge n_B \neq \emptyset), \tag{4.4}$$

$$\text{added/deleted} \Leftrightarrow (n_A \neq \emptyset \wedge n_B = \emptyset) \vee (n_A = \emptyset \wedge n_B \neq \emptyset), \tag{4.5}$$

$$\text{conflicted} \Leftrightarrow (n_A \neq n_B \wedge n_A \neq \emptyset \wedge n_B \neq \emptyset), \tag{4.6}$$

where $n_A \in A$ and $n_B \in B$. Note that red states cannot be resolved automatically. Those nodes that correspond and are equal, are considered unmodified. Since transformations are part of a scene graph

2-way

n_A	n_B	state
$\odot$	$\odot$	unmodified
$\odot$	–	added/deleted
$\oplus$	$\otimes$	conflicted

3-way

$n_{AncestorAB}$	n_A	n_B	state
$\odot$	$\odot$	$\odot$	unmodified
–	$\odot$	$\odot$	
$\odot$	$\oplus$	$\oplus$	
–	$\odot$	–	added
$\odot$	–	$\odot$	deleted
$\odot$	$\oplus$	–	deleted/modified
$\odot$	$\oplus$	$\odot$	modified
$\odot$	$\oplus$	$\otimes$	conflicted
–	$\oplus$	$\otimes$	

Table 4.1: *Schematic representation of a 2-way vs. a 3-way diff. Discrepancies in revisions A and B of the same node n are marked as $\oplus, \otimes$. Since red states cannot be resolved automatically, users' intervention is required when merging. Note that the table is symmetric for n_A and n_B.*

structure in 3D Repo, repositioning a mesh would constitute a change on the transformation matrix but not the mesh itself. However, even if the timestamps of the models were taken into account, they would not necessarily guarantee the temporal ordering of changes. This is because there is potentially a large number of modifications that might have happened concurrently over a longer period than implied by the timestamp. Hence, any discrepancy in the state of two corresponding nodes is considered conflicted. Similarly, a state of a node that is present in only one of the revisions is ambiguous. Although 3D models to grow in complexity over time, see Chapter 6, the node could have been equally deleted rather than added. This can only be reliably determined by their ancestral 3D model as shown in Table 4.1.

4.2.2 3-way diff

Let *AncestorAB* be the previous common revision of A and B, i.e. their ancestral set of scene graph components. The extra knowledge of the original state of the input 3D models helps to automatically resolve some of otherwise ambiguous cases. Hence, the possible states of a component n are as follows:

$$\text{unmodified} \Leftrightarrow (n_A = n_B \wedge n_A \neq \emptyset \wedge n_B \neq \emptyset), \tag{4.7}$$

$$\text{added} \Leftrightarrow n_{AncestorAB} = \emptyset \wedge [(n_A = \emptyset \wedge n_B \neq \emptyset) \vee (n_A \neq \emptyset \wedge n_B = \emptyset)], \tag{4.8}$$

$$\text{deleted} \Leftrightarrow n_{AncestorAB} \neq \emptyset \wedge [(n_A = \emptyset \wedge n_B = n_{AncestorAB}) \vee (n_B = \emptyset \wedge n_A = n_{AncestorAB})], \tag{4.9}$$

$$\begin{aligned}\text{deleted/modified} \Leftrightarrow n_{AncestorAB} \neq \emptyset \wedge [&(n_A = \emptyset \wedge n_B \neq \emptyset \wedge n_B \neq n_{AncestorAB}) \\ &\vee (n_B = \emptyset \wedge n_A \neq \emptyset \wedge n_A \neq n_{AncestorAB})],\end{aligned} \tag{4.10}$$

$$\begin{aligned}\text{modified} \Leftrightarrow n_A \neq n_B \wedge n_A \neq \emptyset \wedge n_B \neq \emptyset \wedge [&(n_A = n_{AncestorAB} \wedge n_B \neq n_{AncestorAB}) \\ &\vee (n_A \neq n_{AncestorAB} \wedge n_B = n_{AncestorAB})],\end{aligned} \tag{4.11}$$

$$\text{conflicted} \Leftrightarrow n_A \neq n_B \wedge n_A \neq \emptyset \wedge n_B \neq \emptyset \wedge n_A \neq n_{AncestorAB} \wedge n_B \neq n_{AncestorAB}, \tag{4.12}$$

where $n_A \in A$, $n_B \in B$ and $n_{AncestorAB} \in AncestorAB$. As before, red states have to be resolved manually. Table 4.1 shows a comparison of a 2-way versus a 3-way diff. Even in a 3-way diff, however, it is the models A and B that are being compared, so if a node is present in *AncestorAB* but neither in A nor in B, this is not considered a difference in A and B.

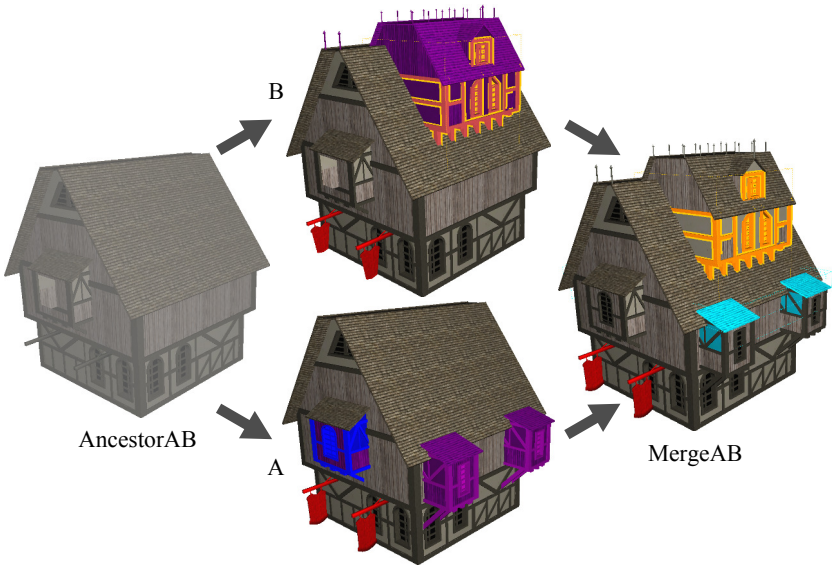


Figure 4.4: *3-way 3D Diff using a common ancestor to resolve conflicts. The 3D Diff UI automatically highlights the differences in two versions A and B of the same model. Inclusion of their common ancestor AncestorAB helps to resolve some of otherwise ambiguous cases. Subsequent interactive user-driven merge resolves remaining conflicts. Modifications are blue, additions violet, explicit conflicts red, implicit conflicts turquoise and user selection orange.*

4.2.3 N-way diff

A prevalent issue described in Chapter 6 is the *"disk full of models"* problem. That is, repositories of 3D models created over a long period of time where the modellers do not even know which models are the modified versions of the same base model. For this reason, the approach of 2-way and 3-way differencing can be generalised to an arbitrary number of 3D models. Same as in a 3-way diff, it is possible to compare every 3D model with all the remaining models in a given set Ω. The number of required pair-wise comparisons is then described as a binomial coefficient $C(n,2) = \binom{n}{2} = n(n-1)/2$, where $n = |\Omega|$ is the number of 3D models being compared as equality is a binary relation.

4.2.4 Sequential diff

Similarly to an N-way diff, it is also possible to perform a sequential diff in order to compare multiple 3D models that evolved over time, although, in a pre-defined sequence rather than a set. Sequential diff, therefore, performs differencing on pairs of 3D models so that the output from one differencing becomes the input to the next, creating a sequence that can be visualised as a 3D editing timeline, see Chapter 6.

4.3 3D Merging

3D merging, depicted in Figure 4.4, can be described as an interactive user-driven process that combines two or more versions of the same input 3D model into a coherent result. The aim is to produce a visually pleasing output that is syntactically and semantically correct and preserves as many of the desired changes as possible. This process can be partially automated depending on the presence of a common ancestor of the input 3D models, see §4.2.2. Given that 3D differencing took place, the merge process can resolve all but the implicit and explicit conflicts. Whenever a node has been marked as one of the

added or *modified*, it must be the latest change, hence can be copied into the merged result. In contrast, a node marked as a *deleted* is left out of the output altogether. If a node is labelled as an *unmodified* in either of its revisions n_A or n_B, it can be copied over. However, explicit conflicts, defined in Section 4.2, cannot be resolved automatically. In such cases, the user has to make the final decision as to which revision is going to be retained and which discarded. Nevertheless, it is desirable to always let the user override any of the automated results. This is because some conflicts might not be resolvable by simply selecting one of the corresponding scene graph nodes. Thus, it is possible to leave the conflicted nodes in the resulting 3D model with the intention to edit such changes in a external 3D editor. Therefore, in conclusion, the automated merge results should always be treated as suggestions only.

4.3.1 Visualisation strategies

Even though some of the later research in the field concentrates on 3D merging as a problem of graph matching between vertices and edges only [DP13], it is argued here that in order to achieve the desired merge results, conflict resolution has to be performed visually in 3D space. Gleicher et al. [GAW+11] formulated a taxonomy of the most prevalent information visualisation strategies as follows:

Superposition. Superimposing several versions of the same scene is a strategy previously used in the differencing and merging of 2D drawings [CWC11]. There, it is easy to notice the differences or flicker between revisions to visually convey changes, e.g. 'Swipe' and 'Onion Skin' image differencing on GitHub [McE11]. However, in the 3D domain, it is difficult to distinguish between individual objects that are rendered on top of each other, see Figure 4.5a for an example.

Juxtaposition. Presenting previews in a side-by-side visualisation offers additional benefits over superposition alone as it places the objects next to each other avoiding visual clutter. However, the user is required to rely more on his or her memory [GAW+11]. This strategy is used in '2-up' image differencing on GitHub [McE11], four-sided view of the same scene in 3D software such as Autodesk 3ds Max [DD13], or in a side-by-side comparison in Vistrails [Cal09], see Chapter 2.

Explicit encodings. Explicit encodings provide direct visualisation of relationships between objects such as substraction or a time warp. Although this type of visualisation pre-computes the results for the user, it is often hard to understand what is being shown due to the missing context. Such representations have been popularised mainly in the field of comparative genome visualisation where the number of potential matches is vast [DMBP04, MMP09]. In 3D Diff, this would translate to showing only the differences without the rest of the input 3D models.

Formative feedback from colleagues as well as professionals experienced in polygonal 3D modelling, some of whom created the sequences presented in Chapter 6, suggests that a suitable approach to 3D differencing is to highlight the detected discrepancies in a side-by-side view and combine the changes into a larger overlay window which contains the current state of the output 3D scene, i.e. the results of the merge process, with conflicting changes being superimposed. This takes the best of all strategies and presents a novel *hybrid* visualisation implemented in §4.4.2.

4.4 Prototype Implementation

Same as the 3D Repo desktop client presented in Chapter 3, the 3D Diff GUI, depicted in Figure 4.5, is written in C++ and a cross-platform framework Qt [BS08], while rendering and navigation are enabled via the GLC Lib [Rib14]. The Open Asset Import Library (ASSIMP) [SGK+14] converts the most common 3D file formats into a unified in-memory scene graph representation that enables comparisons across various file formats and makes this prototype independent of any particular 3D modelling package, see §3.4.3. Nevertheless, the choice of a generic scene graph gives rise to some limitations, most notably the need to rely on the export from 3D editing tools or VCSs such as 3D Repo that might not preserve all of the original data from the editing sessions. Although it would be possible to embed 3D Diff as a plug-in inside the existing modelling packages, some, such as Autodesk 3ds Max, tend to have difficulties rendering multiple 3D contexts within the same application instance. In contrast, a stand alone UI is able to compare many different generic scene types and file formats and can connect to various VCSs.

4.4.1 Scene node equality

In order to determine whether a modification between two corresponding scene graph nodes has happened, 3D Diff establishes whether they are equal or different in any way. As mentioned in Section 4.2, equality of the scene graph components is purely an implementation dependent definition and may be varied from application to application based on the type of data being processed. In 3D Diff, if the same SID is not present in one of the models, it is flagged as deleted since the equality does not hold. However, if it exists in both models, the system performs an early reject byte-by-byte comparison. Currently, the implementation considers only binary changes in meshes, materials and textures, but it can be easily extended to other standard scene graph components such as animations, bones and even shaders. If there are any discrepancies between corresponding nodes, these are labelled as different. This, unfortunately, creates some false positives for meshes that are the same except for polygon order. In general, however, such edits are rare and if they were to occur, it would be visually obvious in the 3D preview.

4.4.2 User interface

The intention of the 3D Diff interface, shown in Figure 4.5, is to abstract away from the underlying differencing technique as much as possible. Therefore, the UI looks the same regardless of it performing a 2-way or a 3-way diff. In a 3-way diff, the ancestral 3D model is hidden from the user by default as it is only used to inform the differencing process as described in §4.2.2. After loading the input 3D models, the user can select which of the *overlay*, *standard* or *smart* visualisation strategy to use. Superposition, also known as an overlay visualisation, shows the differences as well as the merging suggestions in a single window hiding the original models from user's attention. As demonstrated in §4.5.1, this kind of UI is less productive and more prone to error, hence considered only a basic approach to 3D differencing. In contrast, a standard visualisation shows the input models in separate internal windows, while the merged result is shown as a larger preview such that their aspect ratios correspond, see Figure 4.5b. This ensures that in a 3D preview the same proportion of the overall scene is visible. Additional smart option highlights indirect conflicts defined in Section 4.2. Differences are shown as colour-coded highlights

on top of the scene components. In a merge list, the same differences are shown for the user to select which of the two versions to preserve during merging using tick boxes, see Figure 4.5. For conflicting edits, either of the revisions can be selected and the navigation interlinked across all windows. An auto-selection camera reframes the viewport so that the selected object becomes the main focus.

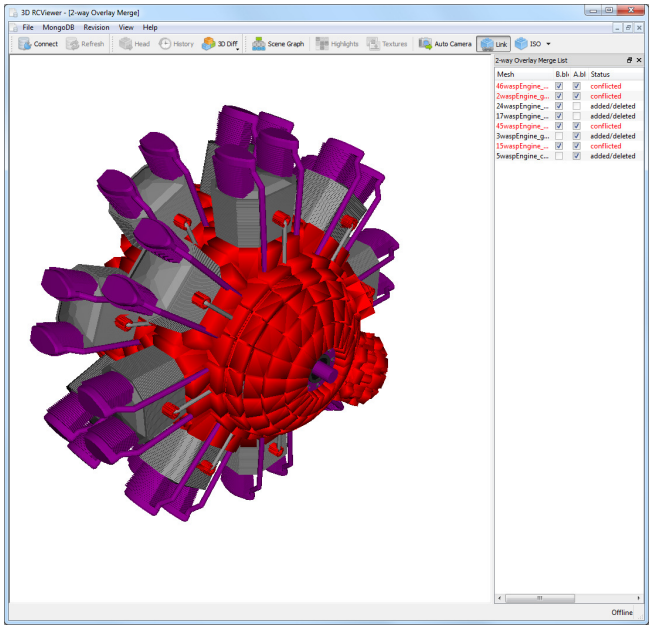
4.5 Evaluation

A prototype 3D Diff tool was evaluated on a number of different 3D scenes varying in size and complexity, some of which are shown in Figures 4.1 to 4.6. The tool was found to work reliably across the different scenarios providing useful insights into the editing provenance as well as creating consistent merge suggestions especially with the inclusion of an ancestral 3D model in a 3-way diff. In order to determine the best approach to visualisation of 3D differencing and merging, and to evaluate the amount of trust the users are likely to attribute to each of the techniques, a pilot user study with eight participants was undertaken. All participants were postgraduate students in the fields of computer graphics or vision with a medium to high level of experience in polygonal 3D modelling. The study tested 3D merging of pre-made changes with an increasing level of software assistance. This provided a unique opportunity to obtain an early feedback on the prototype implementation of the 3D Diff UI.

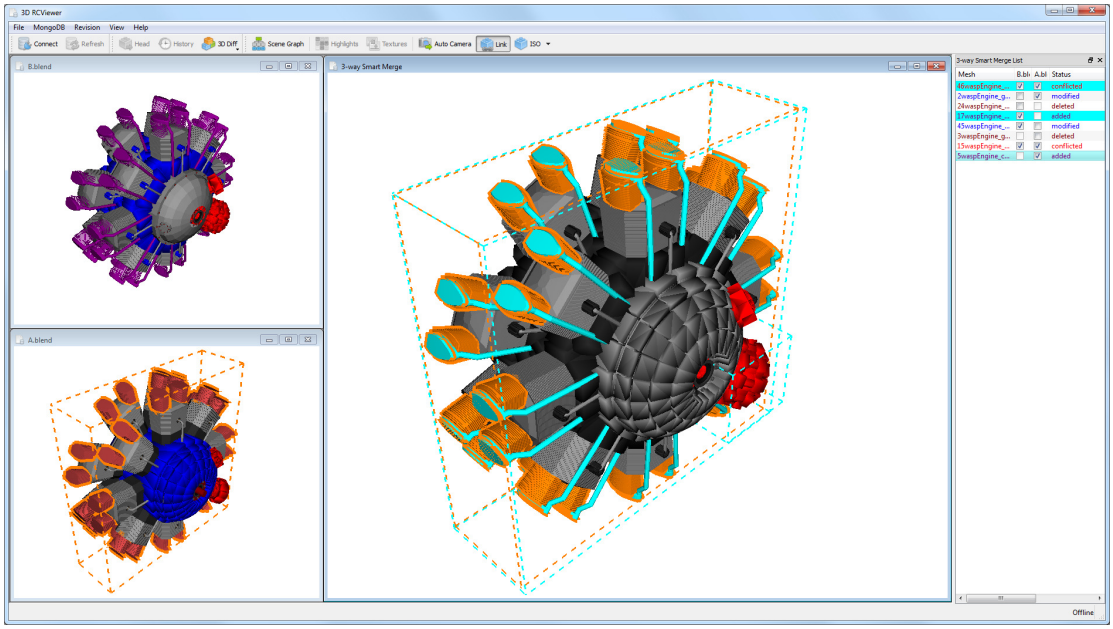
4.5.1 User study

In each trial, the participants had the same task of merging changes in 3D models so that the most recent versions of the meshes were to be preserved while not introducing any new implicit conflicts. If conflicts could not be resolved using the 3D Diff alone, they were asked to leave the offending components in a conflicted state and indicate that they would rather use an external editor to resolve them manually. After each task, the participants were further instructed to fill in an electronic questionnaire to comment on their experience and at the end of the whole experiment also to rate the techniques from the best to the worst, see Appendix C. Each participant completed all of the T_1 2-way diff overlay, T_2 2-way diff standard, T_3 3-way diff standard and T_4 3-way diff smart scenarios with different sets of 3D models. Each dataset M_1 to M_4 consisted of a sample 3D model such as a robot or an engine with less than 100 distinct components, e.g. Figure 4.5, and a large 3D scene such as a city or a castle with more than 100 components, e.g. Figure 4.6. The scenes exposed the participants to all five possible states a component might be in, that is *unmodified*, *added*, *deleted*, *modified* and *conflicted*. Each small model had eight of such states, two per category, while each large model had twice as many. The order in which the participants performed the tasks as well as the datasets where shuffled according to Latin square in order to suppress any effects of learning.

As shown in Figure 4.7, the highest average scores for ease of use, no technical difficulties and trustworthiness were achieved by a standard 3-way diff, although, on average, the participants trusted automated suggestions more as the level of software support increased, ranking the 2-way overlay the least, while the the 3-way diff with an implicit conflict detection being the most trustworthy. Based on the participants' written comments it is believed that the implicit conflict detection is a useful addition to the system, although, in its current implementation where multiple meshes belong to a single component,



(a) *2-way overlay*



(b) *3-way smart*

Figure 4.5: *2-way overlay vs. 3-way smart visualisation. A stand alone 3D Diff GUI loads two versions of the same 3D model in order to highlight discrepancies amongst them.* **(a)** *In a 2-way overlay visualisation, only the merged result is shown.* **(b)** *In a 3-way smart visualisation, the two versions that are being differenced are shown together with the merged result. Indirect conflicts are in turquoise colour.*

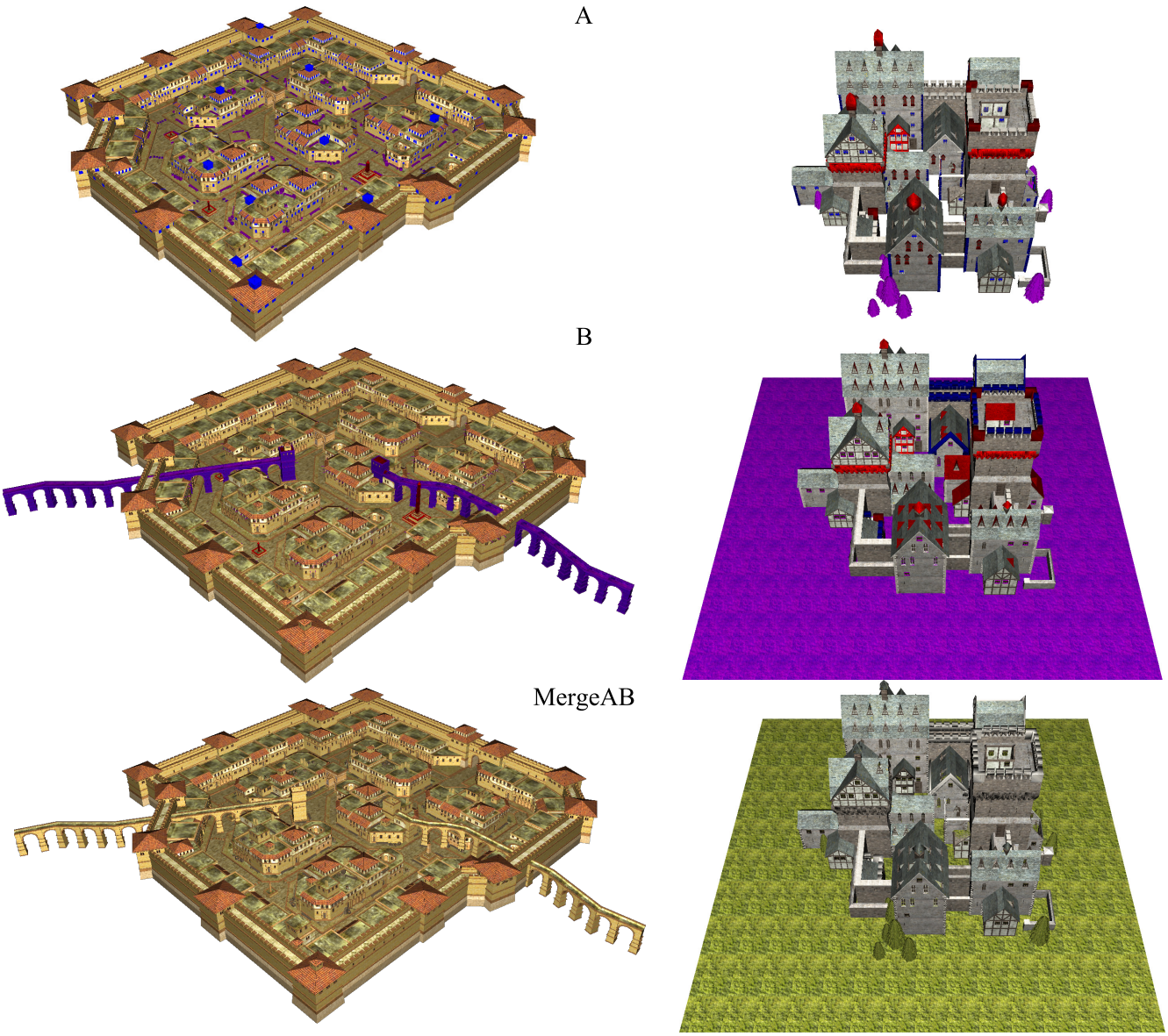


Figure 4.6: *Examples of large 3D scenes used in a pilot user study. A and B are the individual revisions, while MergeAB the combined result with conflicting edits resolved. Models courtesy of Johnathan Good.*

large bounding boxes would indicate a conflict despite no apparent intersection with another mesh. As one participant noted: *"The conflicted BB [bounding box] is useful but needs to be calculated at lower levels of meshes."* Similar comments were made by others, too.

4.6 Discussion

The two main contributions of this chapter are in identifying the problems of 3D differencing and merging, and in presenting a prototype tool that supports interactive conflict resolution as described in Sections 4.2 and 4.3. Since 3D Diff decomposes the most common 3D file formats into their scene graph components before differencing, it ensures that it is independent from any one modelling software. Actually, this technique enables differencing of otherwise incompatible 3D file formats. The stand alone model viewer performs an early reject byte-by-byte comparison of the scene graph nodes from various

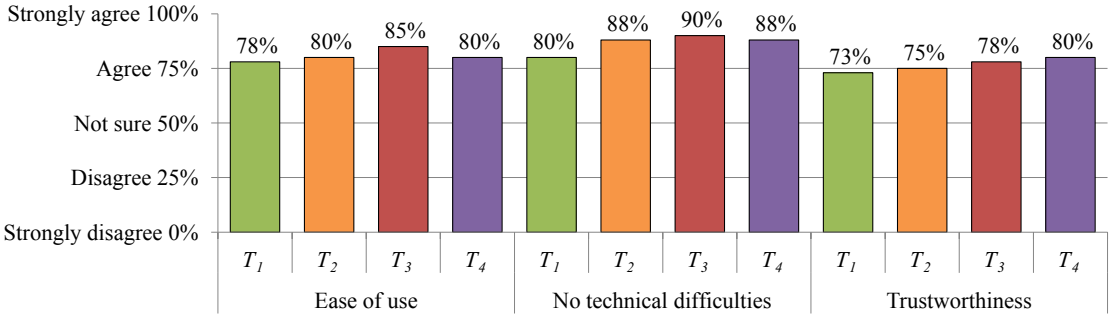


Figure 4.7: *3D Diff user study questionnaire results based on sample averages. T_1 to T_4 correspond to tested visualisation techniques with an increasing level of software support for merge suggestions.*

revisions that share the same SID. When there are discrepancies in the two models, these are treated as conflicting edits. However, a 3-way Diff is able to identify six instead of the basic three states, some of which can be resolved automatically. If one of the revisions is the same as the original, the other must be the intended modification.

Participants of the user study in §4.5.1 were able to use the prototype successfully and were able to explore different choices that the UI provided, even though they varied in what they considered to be a successful merge. Some conservatively discarded changes, whilst others, for aesthetic reasons, reverted additions or deletions that were flagged automatically. Thus, perhaps in a way that is not apparent in software merging, 3D merging is a subjective judgement. Hence, the future tools should not assume that the chronology of edits is the primary selection mechanism but should focus on the aesthetic justification as well. Based on this initial evaluation, it is believed that such an approach to 3D conflict resolution can significantly speed up the revision control management of polygonal 3D models in 3D Repo. Although it has not yet been explored, the 3D Diff interface could be connected to other VCSs such as SVN. Nevertheless, the user still has to confirm all changes before applying them as otherwise undesired modifications violating semantic relationships might be propagated. Integration into modelling packages via their plug-in frameworks was not performed in this thesis but is certainly possible, see Chapter 7.

4.6.1 Limitations

Differencing and merging of 3D models presents new challenges in computer graphics. Because specific decisions about the granularity of the tracked changes were made, the current 3D Diff prototype supports only comparisons between nodes in a scene graph. In the future, it would be possible to extend this approach to take into account more fine-grained modifications to provide a truly robust solution for 3D asset management. For example, based on the established node differences, other algorithms could compare individual vertices and find only those portions of the meshes that have been modified. Also, the system does not support any changes in the structure of the scene graph. If the scene graph layout has been modified significantly, i.e. a node or an entire branch have been reparented, 3D Diff would replace the modified chain of scene node components regardless of any structural impact. However, in some cases, it might lead to visually unpleasant results and these would have to be resolved manually.

Currently, the 3D Repo framework utilises universally unique identifiers (UUIDs) [Tel08] for the SID values that define a node-to-node correspondence in 3D Diff. Such 128-bit numbers can be repre-

sented as 32 hexadecimal characters, plus four hyphens that can be omitted. Nevertheless, many 3D file formats impose name size restrictions that are smaller than that. Autodesk 3ds file format, for example, supports only 10 characters, while Blender blend format has only recently increased its limit from 24 to 64 characters. To overcome such problems when importing polygonal 3D models, 3D Diff hashes individual name strings into UUIDs. All that remains to do is to ensure that modellers when creating new scene graph nodes did not introduce the same names concurrently. The simplest approach is to agree on some kind of naming convention, for instance, pre-pending each new component with unique initials of each of the modellers. Of course such an approach would not be applicable to legacy datasets that might not have followed any set conventions in which case a more robust correspondence detection will be required. Chapter 6, therefore, introduces a novel matching algorithm which is independent of any data structures in the input models and is demonstrated to work well on large datasets composed of hundreds of 3D files and many millions of polygons in a matter of seconds.

4.7 Chapter Summary

Chapter 3 introduced a novel version control framework specifically designed for 3D assets in order to partially address the first research question defined in Chapter 1. The outstanding question was whether it is possible to sustain collaborative 3D editing without the need for asset locking. This chapter, therefore, introduced a highly related concept of visual 3D differencing and merging. This is becoming increasingly important as it is poorly addressed by the current visualisation tools and even 3D editing packages. In framing the problem, analogies to source code management were made. Such a comparison is especially interesting as a vast range of research has been performed on software merging and a number of different tools have been built over the years. Unfortunately, processing pure textual changes on 3D models does not work as the semantic structure of the files would not be preserved. On the other hand, administering differences on binary assets suffers from similar issues due to the diverse nature of 3D models and the types of data representations that constitute a complex scene. Thus, in Section 4.1, a scene graph has been chosen as an abstraction layer which fits well into the 3D Repo version control framework. This ensures that the process of 3D differencing can be fully automated so that it generates syntactically and semantically correct merge results. Apart from explicit conflicts, where the same parts of a 3D scene were edited concurrently, the potential of implicit conflicts due to violations in semantic meaning was considered. It is, therefore, important to follow the rules of the design and construction when merging changes by multiple authors. Those changes that introduced bounding box intersection where there were none before are labelled as implicit conflicts.

To demonstrate the proposed visual 3D differencing and merging, a prototype tool, 3D Diff, was developed in Section 4.4, and evaluated in a pilot user study in Section 4.5. This novel interactive interface performs 2-way and 3-way differencing and merging between various types of 3D scenes. The input to the system are two polygonal 3D models and, optionally, their common ancestor, that are to be compared by their scene graph nodes. 3D Diff calculates differences down to polygon meshes, materials and textures stopping at the point when the corresponding objects are different. It then provides options for resolving differences by allowing the user to select which parts of the conflicted 3D models to retain

from which input. The output may not be completely resolvable because components might be explicitly conflicted or new implicit conflicts might have been introduced by the merging process itself. In such cases, the partially resolved merge result is exported as a new 3D model and modified in a vertex-level editor. Further contributions include the development of multiple visualisation modes for both 2-way and 3-way differencing combining approaches such as superposition, juxtaposition and explicit encodings.

A formative user study indicates that the users of 3D Diff are able to perform merging tasks efficiently, and can exploit the advantages of a 3-way diff including highlighting of potential implicit conflicts. This suggests that the tool is promising and likely to be useful in everyday version control tasks of differencing and merging. However, it currently relies on a known node-to-node correspondence via SID values as assigned by the 3D Repo system in Chapter 3. In situations where such revision metadata based on unique scene node identifiers is missing, this differencing tool would not be applicable. Thus, the conclusion is that a reverse engineering solution for legacy datasets that were recorded outside of a VCS is still required. To address this limitation, Chapter 6 introduces a novel correspondence estimation algorithm that makes no assumptions about the scene structure or metadata and provides a unique timeline interface in order to visualise the provenance of a sequential differencing.

Chapter 5

XML3DRepo Daemon Service

The research presented in this chapter was undertaken in collaboration with Kristian Sons and Dmitri Rubinstein supervised by Prof. Philipp Slusallek in the German Research Centre for Artificial Intelligence (DFKI). As the first author of this work, my personal contribution spans the definition of a novel REST API in Section 5.2, development of a server-side daemon service retrieving data from the 3D Repo repository in Section 5.4, the experimental evaluation of various web browsers and encoding formats conducted in Section 5.5 as well as the discussion covered in Section 5.6. The full results of the collaboration are included for completeness.

Chapter 3 introduced a novel system for management and visualisation of non-linear history of polygonal 3D models and Chapter 4 addressed the issues of 3D differencing and merging. However, the second research question in Chapter 1 further asks whether such a specific 3D versioning framework can deliver real-time visualisations that would be independent of the underlying data store. As described in Section 3.6, the applications that connect directly to the remote repository exposing the contents of the underlying data store. By design, NoSQL DBs do not validate schema on insertion. Since no changes have been made to the DB itself, the version control logic had to be developed at the application level. Each client is, therefore, required to adhere to the principles of 3D versioning, e.g. not removing any older revisions, ensuring presence of revision metadata defined in §3.2.2, etc. Unfortunately, as soon as any third-party client connects to the system, the integrity of the repository cannot be enforced any more.

This chapter, therefore, introduces a *daemon service* XML3DRepo which acts as a layer of database indirection on the server-side in order to address this limitation. The aim is to prevent raw data access and to provide a form of repository safeguarding via a gateway service. Hence, a novel REST API is defined in Section 5.2 which supports 3D version control but does not expose the DB connection to the clients. A prototype implementation of the proposed system architecture including a novel web browser-based client is presented in Section 5.4. This supersedes the original web client from §3.4.4 not only in functionality but also in its ability to run on any WebGL-enabled platforms including mobiles and tablets, not just desktops. Figure 5.5 serves as an example. Efficiency of the prototype was tested using multiple data encoding formats as explained in Section 5.5. Based on this evaluation, Section 5.6 concludes that the best data format can be selected only by considering both the network properties and the processing capabilities of the receiving client.

5.1 System Overview

Although originally designed for sharing of text documents, the Internet has become a graphical environment ever since the famous proposal for the <IMG> tag by Marc Andreessen in 1993 [Pil10]. Since then, the software surrounding the HTML and the HyperText Transfer Protocol (HTTP) has evolved to support all sorts of graphical content including 2D images and polygonal 3D models. The recent boom in online 3D adoption can most likely be attributed to the introduction of WebGL [Mar11] and its wide-spread support in modern web browsers. This lead to the development of countless JavaScript-based 3D libraries such as X3DOM [BEJZ09], GLGE [Bru09], XML3D [SKR+10], SpiderGL [Ben10], SceneJS [Kay10], Babylon.js [CRLR13] and arguably the most popular of them all Three.js [Cab10]. This has exerted further pressure on 3D content creation with the desire for even more complex scenes being manipulated over the Internet. Web based tools are becoming ever more important in the shift from single user 3D desktop applications to shared social experiences online. Nevertheless, the current web 3D technologies are not yet fully utilising modern design patterns for accessing online resources such as REST [Fie00]. The prevalent paradigm in the 3D domain is to use the web as a publishing platform but not a development one. Formats such as Virtual Reality Modeling Language (VRML) [CBM97] and its successor X3D [JIS+13], see Section 2.4, are designed to be updated in place at runtime but do not provide means of persistent modification preservation of their own. In contrast, Wikis [Klo06] and Google Docs [Vie09] are just a few examples of successfully exploiting the web as an editing platform. Yet, an often overlooked property of HTTP is its definition of several verbs suitable for *creating*, *updating* and *deleting* shared resources, not just *requesting* them. With the ever increasing interest in mobile devices, the limiting factors of bandwidth and latency are reintroduced as serious risks, too. Hence, it is important to build an easily accessible and scalable platform for online 3D versioning with support for WebGL-enabled web browsers. Unfortunately, standard data integrity and data exchange tools cannot be used for this task as these either deal with assets at an inflexible per file level or are bound to specific data formats. What is more, a server-side daemon service is necessary to address the main limitation of the system already proposed in Chapter 3. 3D Repo, as the framework is called, enables non-linear management of concurrently edited 3D assets. The versioning functionality including branching and merging is built atop of a centralised NoSQL database MongoDB [MPH10]. However, the prototype implementation developed in Section 3.4 supports only raw database access through BSON queries. Such a direct repository connection cannot ensure data consistency, a problem that was already identified in Section 3.6. Thus, the aim of this chapter is to provide seamless data encoding where 3D assets are independent of their fixed file formats, and can be accessed in a consistent way over the Internet. This novel platform would enable ubiquitous access to 3D data in a form most preferred by the receiving client. To achieve these goals, a simple yet powerful API that combines persistent version control with the convenience of a REST paradigm is proposed in Section 5.2. Unlike other approaches described in Chapter 2, this API enables client applications to mix and match different encoding formats as well as access the previous revisions. This makes the proposed solution a truly file format-independent 3D version control service as files, according to the REST principles, see §5.1.1, become temporary data representations.

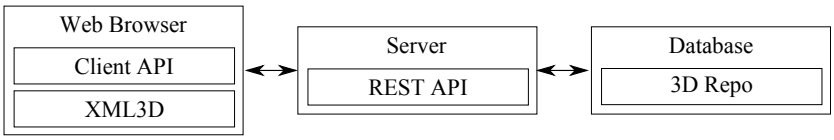


Figure 5.1: *XML3DRepo high-level overview. Client connects to a server using REST API. 3D content is dynamically fetched from the repository and delivered to the web browser running an XML3D client.*

5.1.1 Representational state transfer

Representational State Transfer (REST) [Fie00] is a style of software architecture for the web governing the behaviour of clients and servers in terms of requests for resources and the corresponding responses. In this architecture, a resource is a uniquely addressable piece of data using the uniform resource identifier (URI) string. Its representation is determined by the document which the data is returned in. A closely related concept of persistent storage management, as is the case of a 3D repository, is formed by the functions *create, read, update and delete (CRUD)*. When deploying a web-based architecture, a RESTful API maps all of these methods to the individual HTTP 1.1 protocol verbs [FGM+99] in order of `POST`, `GET`, `PUT` and `DELETE` respectively, hence being "full". In addition, it has to specify the base URI for the web service and the media content type of the supported representations, also known as the Multipurpose Internet Mail Extensions (MIME) [FK05]. Therefore, the only required information for a client to interact with a server is the location of the resource and the intended action. Requesting a document representation makes this style of programming independent from the underlying storage.

5.1.2 System architecture

The purpose of the new REST specification is to define a transparent API that supports a unified access to version controlled 3D resources over the Internet. At the same time, it is meant to support data retrieval that is independent of the enabling technologies or the data store itself. Relying on a RESTful architecture has the further benefit of flattening the scene and revision history DAGs of 3D Repo. This is because the resources can be queried individually using their unique URI, or together as collections of resources based on their common type, also referenced by a URI. As shown in Figure 5.1, XML3D was selected on the client-side for the purposes of a prototype implementation presented in Section 5.4. Even though the rendering could be accomplished by other suitable libraries, some of which are mentioned in Section 5.1, XML3D provides many benefits over alternatives especially in terms of external references to resources, see Section 5.3. In short, XML3D is an extension to HTML5 that defines interactive 3D graphics as part of a web page source code. Similarly to HTML, all XML3D elements belong to the document object model (DOM) [Mar02], a tree representation of the objects in web documents. Thus, they can be easily accessed and modified via JavaScript by attaching events such as `onmouseover` to scene objects. In order to demonstrate the advantages of the proposed solution, several file transfer strategies have been implemented and evaluated for their speed of delivery and encoding efficiency in Section 5.5. These include Extensible Markup Language (XML) [BPSM+08], JavaScript Object Notation (JSON) [ECM13], Binary JSON (BSON) [Mon14a], Sequential Image Geometry (SIG) [BJFS12], Open Compressed Triangle Mesh (OpenCTM) [Gee09] and ArrayBuffers [Khr13].

5.2 Application Programming Interface

3D Repo repository is based on a NoSQL DB which stores individual 3D components alongside their associated revision histories such that each component is treated as a delta change. Due to the DB being the persistent storage provider, 3D Repo avoids constraints of a file-based system and offers extensive querying functionality. To achieve this, every DB entry regardless of it belonging to a scene or a revision, has to specify a revision metadata tuple as defined in Section 3.2. This consists of a unique identifier (UID) which is a functional requirement of a DB, as well as of a shared identifier (SID), which is shared across instances of the same logical document. In a scene graph, the SID represents the same scene component being tracked over time. In a revision history, however, the SID defines a single branch and the SID of all zeros is reserved for the trunk/master. The API outlined in this section, therefore, provides access to 3D resources stored in a 3D Repo repository without exposing the data store. The overarching design can be summarised as a dual URI encoding where in its core lies a combination of the `id` and `type` variables, the order of which determines the behaviour of the interface. In general, to address a collection of resources the `/:id/:type` schema is used, while the inverted `/:type/:id` combination addresses a single resource. Depending on the context, the ID can be either the UID or SID of a resource. In addition, the type variable defines the family of resources such as materials, meshes, textures and even revisions. What is more, each resource or a collection of resources can be requested in various encodings specified via a HTTP header or a format extension at the end of the uniform resource locator (URL), e.g. 'xml', 'json', etc. Such an approach decouples the storage implementation from the querying interface. This successfully resolves the need for a direct DB connection of the original implementation presented in Section 3.4 so that third-party applications can connect to 3D Repo.

5.2.1 POST

Posting data to the server is used to create new repositories as well as to commit new revisions and to perform merges. Here, the UUID is assumed for both the UID and SID.

`/xml3drepo` Creates a new empty repository with a unique name if it does not already exist. Hence, a name string input is expected. Note that when the name is missing or an existing name is submitted, the system throws a suitable error message, see §5.2.6.

`/xml3drepo/:name` Commits a new head revision to the trunk/master of the repository identified by its unique name. This has the same effect as `/xml3drepo/:name/00000000-0000-0000-0000-000000000000`, since the all zeros SID is reserved for the main development path as described previously. The expected input is the data to be committed as well as a new revision metadata such as the revision author, commit message, etc. These are then appended to the revision history DAG with the addition of a server-side timestamp.

`/xml3drepo/:name/:id` Commits a new revision but to a branch identified by its SID. If the desired branch does not exist, it is created. When merging, the posted revision document specifies the revisions to be merged as its parents in the DAG hierarchy of the revision history. The SID is then the one of the branch which lives on after a successful merge operation.

5.2.2 GET

Retrieving data is the most commonly used feature of any such an API. Therefore, the specification has to be flexible enough to provide suitable means of addressing collections of resources, single resources and even individual attributes, i.e. sub-parts of the resources.

`/xml3drepo` Returns a collection of available 3D repositories, i.e. a list of all unique names of databases containing 3D scenes with their revisions.

`/xml3drepo/:name` Returns the head revision of a trunk/master, i.e. all components of a scene identified by its unique name from the latest revision of the main development path. Similarly to `POST`, this has the same effect as `/xml3drepo/:name/00000000-0000-0000-0000-000000000000`. Note that the scene is returned as a flat collection of documents that has to be reassembled by the receiving client into an actual DAG before rendering.

`/xml3drepo/:name/:id` Returns a full scene similarly to the trunk/master's head, but from any revision identified by its unique name and UID. If an SID is submitted instead, this call returns the head of a branch the SID belongs to.

`/xml3drepo/:name/:id/:type` Returns a collection of resources matching the requested type that belong to a revision identified by its UID directly or by an SID if the head of a branch is required. If the type is 'revisions', returns a revision resource describing the author, message, etc.

`/xml3drepo/:name/:type` Returns a collection of resources matching the requested type from the trunk/master's head. If the type is 'revisions', returns the entire revision graph as a flat collection of documents. Similarly to a 3D scene, this collection has to be reassembled into an actual DAG on the client-side before use.

`/xml3drepo/:name/:type/:id` Returns a single resource matching the requested type and the UID which can potentially belong to multiple revisions. If an SID is supplied instead, returns a collection of revisions of the same scene component or a revision history depending on the type.

`/xml3drepo/:name/:type/:id/:attribute` Rather than the entire resource, this request returns a single attribute of the resource. This functionality is useful for non-standard formats such as various Sequential Image Geometry (SIG) encodings that consist of 8-bit sections [BJFS12].

5.2.3 HEAD

The `HEAD` request is defined in the exact same way as `GET`, however, it does not return the actual body of the data. This is especially useful for accessing the DAG structures of a scene graph or a revision history without the need to fetch the contents of any heavyweight components such as meshes, textures or in the case of a revision history the long lists of `added`, `deleted` and `modified` SID values as specified in Section 3.3. Same as in `GET`, the DAGs have to be reassembled locally.

5.2.4 PUT

Idempotence of `PUT` guarantees that sending a request multiple times has the same effect as sending it only once. However, as soon as a resource becomes stored in a version controlled repository such as 3D Repo, it can never change, only become superseded by a newer revision of itself. Alternatively, it can become deleted but also only as a new revision. Thus, a VCS cannot support in place updating. Instead, when using `PUT`, XML3DRepo will commit a new revision to the repository, although, posting all new resources in a single commit is preferred instead of sending them one by one. In addition, this command can be utilised for requesting changes in the state of the repository and resources, e.g. locking. Nevertheless, other APIs listed in Chapter 2 do not support `PUT` requests at all. Therefore, it is more of an implementation decision as whether to enable this type of messaging since the `POST` command can achieve the same functionality more conveniently. Note that 3D Repo does not support locking yet.

`/xml3drepo/:name/:id/:type` Commits a new revision of all resources identified by their type to a branch identified by its SID. If the client is able to define unique revision identifiers, e.g. in 3D Repo the UUID is used, the system can accept a new single revision as well. If the type was 'locks', it would acquire a lock on a revision or a branch and would return a confirmation.

`/xml3drepo/:name/:type/:id` Commits a new revision of a specific resource identified by its UID to the trunk/master or that of a head revision if identified by an SID. If the UID is not at its head revision, the request fails with a conflict. If the type was 'locks', it would acquires a lock on a single resource or all instances of the same logical component if an SID was supplied.

5.2.5 DELETE

As explained in Section 3.3, when deleting data from a version controlled repository, it cannot be actually removed from the data store but merely replaced by a new revision where it is marked as deleted. Thus, the resource can still be accessed via older revisions when required. Similarly to `PUT`, it is preferred to submit all deletes in a single query to avoid unnecessary new revisions being created one by one.

`/xml3drepo/:name/` Force removes a repository identified by its unique name from the database. Unlike deletion of a single or multiple resources, this operation cannot be reverted as the corresponding collection is dropped. Hence, such a query should be used with caution.

`/xml3drepo/:name/:id/:type` Commits a new revision to a branch identified by its SID where all resources identified by the specified type are marked as deleted. If the type was 'locks', it would release the lock from a resource if UID or a branch if SID was used.

`/xml3drepo/:name/:type/:id` Commits a new revision where the resource identified by its UID, or SID if the head is required, is marked as deleted. If the type was 'locks', it would release the lock from the resource.

To overcome the missing `DELETE` functionality of some web browsers, it is customary to generate a hidden field value 'delete' and use `POST` or `PUT` instead [RR07]. When receiving a form with this parameter, the server overrides the actual HTTP request and fulfils the desired action.

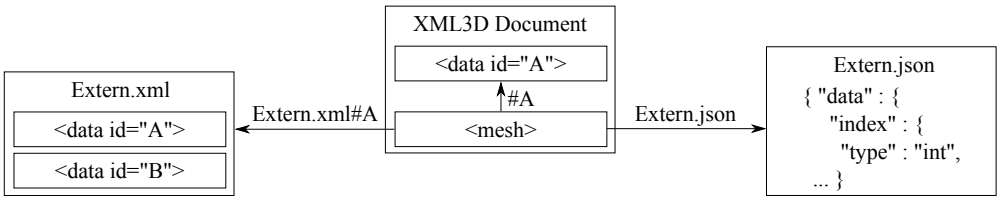


Figure 5.2: *Different ways of referencing resources in XML3D. Intra-document resources (#A), entire document as a single resource (extern.json) and resources inside external documents (extern.xml#A). This mechanism applies to all types of resources such as shaders, animations, etc., and not just meshes as depicted in this example.*

5.2.6 Status codes

Various HTTP/1.1 status codes can be received when using the API. The most commonly used ones are the `400 Bad Request` when invalid syntax was submitted, `201 Created` when successfully committed, `406 Not Acceptable` when a requested encoding could not be achieved and `409 Conflict` when committing changes that are in conflict with the head revision. In this case, a list of conflicting entities should be returned, see [FGM+99].

5.3 XML3D

XML3D by Sons et al. [SKR+10] is a modern *declarative* scene graph representation defined as an extension to the base HTML5 specification. Another example of a declarative 3D technology is X3DOM by Behr et al. [BEJZ09] which together with XML3D forms the evaluation platform of the World Wide Web Consortium (W3C) Community Group *Declarative 3D for the Web Architecture*. X3DOM specification is based on X3D which in turn is based on VRML97 and, therefore, inherits many of its original concepts. Although X3DOM supports many features that are similar to XML3D, the latter was chosen due to its consistent data handling capabilities. As explained in Figure 5.2, XML3D supports multiple ways of referencing external resources. Such a functionality is crucial for the experimental evaluation of the proposed API presented in Section 5.5 as it enables comparison of several different methods of data delivery from the server to the client.

According to Behr et al. [BJFS12], only about 5% of the overall polygonal 3D model size can be attributed to the scene graph structure including transformation groups and materials. The remaining 95% consists of heavyweight unstructured data containers such as meshes and textures. Since declarative 3D by definition embeds scenes directly into web pages, it is necessary to externalise as much of the model information as possible. This is because in general, web browsers will not render a page until the whole HTML DOM definition has been loaded. However, at the same time, the web browsers are very good at asynchronously fetching data linked via external containers that can significantly improve the responsiveness of the system. In human-computer interaction (HCI), it is well-known that if an application does not provide feedback within 1 to 10 seconds, the user will lose interest or consider the application broken [Nei94]. Hence, it is crucial to load large visual 3D data containers independently of the actual web page in order to provide a suitable user experience.

5.3.1 Data referencing

One of the core components of XML3D is the `<data>` element. This element provides functionality that groups arrays similarly to the *buffer* data structures of modern graphics APIs. Unlike buffers such as vertex arrays, however, these elements can reference other `<data>` elements creating nested, recursively defined structures. Figure 5.2 shows three different ways of referencing that are possible with XML3D. Firstly, references can point to resources within the same document, also known as *intra-document resources*. These are suitable for 3D scenes that are small in their number of components and data sizes. Secondly, references can point to external resources defined in a single file. Such an approach is suitable for representations that cannot be natively accessed from within the DOM definition itself but have to be parsed using JavaScript. Although only a single scene graph component can be accessed in this way, this already enables web browsers to fetch data asynchronously just like when loading externalised images or cascading style sheets (CSS). Regardless of the increase in loading speed, which is bound by the available bandwidth, a much more important side effect is the increase in perceived responsiveness of the web page itself. Instead of waiting for the whole 3D model to load, the renderer can now visualise individual scene components as they become available. This is possible only because the whole scene graph structure including world-space transformations is already pre-loaded as part of the HTML page. Finally, multiple resources that reside within the same external document can be referenced simply by using standard URI semantics. This enables grouping of scene entities into larger data blocks especially where these would be too small or inefficient on their own, e.g. materials. These different concepts together with the URI referencing of the proposed API provide a remarkably fine-grained control over the composition of large 3D scenes that are to be loaded from a remote repository such as 3D Repo. In this system, data can be reused as well as organised across multiple resources if necessary. Such data concepts apply to `<mesh>` elements that define geometry of a 3D scene, `<shader>` elements that describe material properties as well as `<lightshader>` elements that describe available lights. All of these are just specialised `<data>` elements.

A declarative dataflow extension to XML3D is *Xflow* by Klein et al. [KSR+12]. In addition to the core definition, each `<data>` element can reference an operator that computes an output from entries of a data block which act as its input parameters. Therefore, Xflow is able to transform `<mesh>`, `<shader>` and `<lightshader>` elements into sinks of a dataflow. It thus provides functionality for dynamic mesh morphing, animation of shader parameters, etc. Since in this way XML3D supports external references to arbitrary data containers without the need for any further extensions, it is possible to implement the delivery formats as proposed in §5.1.2. To achieve this, the only required action is to develop a loader plug-in for each data format independently to decode the resources and to map them onto collections of data entries in the base XML3D scene graph. This even enables the client to mix and match several different representations in order to recompose a single 3D resource. In contrast, the *Inline* node mechanism of the VRML/X3D definition provides means for inclusion of complete subsecenes only. Inline node, therefore, cannot be used for a fine-grained referencing as there is no way to access parts of a subscene [BJFS12]. Hence, X3DOM is not suitable for the work described in Section 5.5.

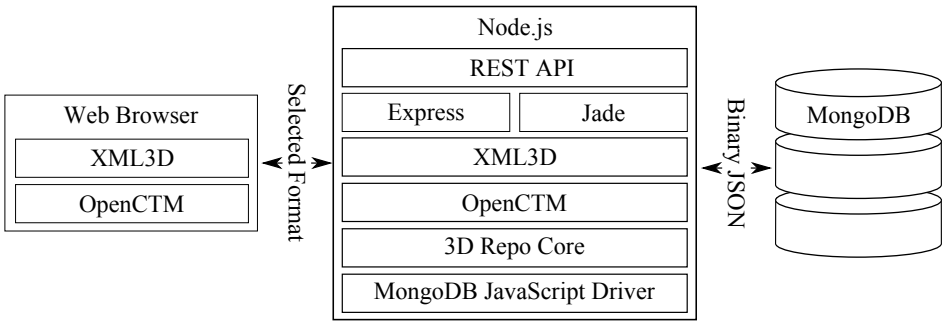


Figure 5.3: *XML3DRepo prototype implementation overview. Remote web browser client connects to a server using the REST API. Dynamic web pages are created on demand using the Jade templating engine and Express middleware. Unlike 3D Repo, this system enables data retrieval in a variety of encodings.*

5.4 Prototype Implementation

A prototype implementation of the REST API was developed in XML3D and *node.js* [Ihr13]. As explained in Section 5.3, XML3D was chosen because it enables fine-grained data compositing when rendering 3D models on the web. Node.js, on the other hand, was chosen because it has gained a lot of attention recently due to its non-blocking asynchronous event handling capabilities. This framework is based on JavaScript that is executed in Google's run-time engine V8 [Goo08b] that was originally written for their Chrome web browser in C++. This is especially useful as it enables even inexperienced web developers to create scalable server-side applications without the need to learn yet another scripting language such as PHP [Tat13] or Ruby on Rails [FF14]. It, therefore, bridges the server and client development so that both sides can share the same basic principles, programming environment and common data formats. What is more, traditional web servers such as Apache HTTP Server [BC08] designate a new thread to each incoming request creating an unnecessary overhead due to the required memory allocation and context switching. For the purposes of 3D version control this would be highly limiting as there are often thousands of 3D components that have to be queried independently. Multiplying that by a large number of concurrent users would almost certainly cause the server to become unresponsive. In contrast, node.js uses a single thread and relies purely on asynchronous callbacks for scalability [Rau12].

Figure 5.3 depicts the overall system architecture and software components that were used to develop the prototype. Middleware *Express* and templating engine *Jade*, both available as internal node packaged modules (NPM), were used to provide the API routing mechanism and HTML output definition respectively. In addition, calls to the OpenCTM C++ library were executed directly from JavaScript. However, instead of relying on the original 3D Repo Core library as defined in §3.4.3, a new JavaScript port was developed in order to utilise the callback functionality and scalability of node.js. Same as in 3D Repo, the UID and SID revision metadata were realised as UUID [Tel08] values. Therefore, any resource can be directly addressed via its UID, e.g. `/xml3drepo/UT4_Baeza/meshes/4e992f02-3777-41ad-b777-91ad377791ad.xml`, although strictly speaking, the format extension, such as 'xml' in this case, can be omitted. This is because the HTTP Accept header sets the desired format in the request [FGM+99]. Most of the time, however, it is present or specified as a query parameter such as `?meshformat=xml`. If the header and format are in conflict, the header gets a preference.

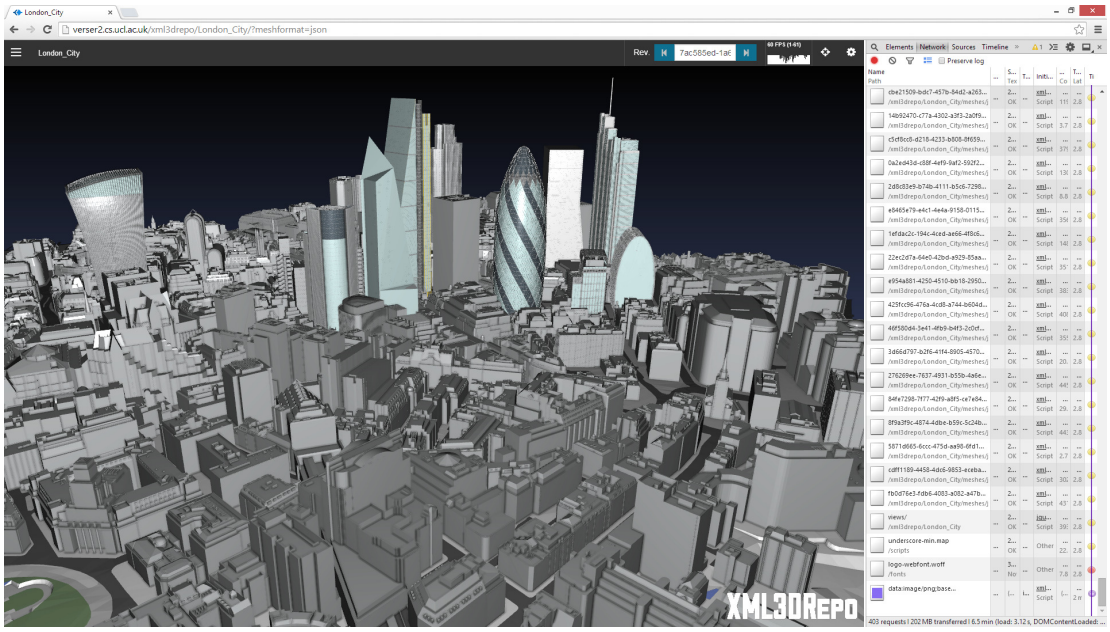


Figure 5.4: *XML3DRepo in Google Chrome web browser on desktop. Visualised is the square mile of London 3D model which is accurate down to 150 mm in real world. Right hand side shows the Chrome Developer Tools listing individual API requests being sent from the browser to the server. Model courtesy of Vertex Modelling.*

5.4.1 XML3DRepo web client

The associated web client, shown in Figure 5.4, was developed in XML3D and JavaScript. Its purpose is to replace the original web client presented in §3.4.4 with support for the new API. Although declarative 3D has been proposed as an extension to HTML5, it has not been officially ratified, yet. Therefore, XML3D does not belong to the standard W3C technology stack recommendation. Nevertheless, this specification is already supported in two different implementations. First is based on the native web browser modifications that have been forked as separate projects. There is the support for Mozilla browser framework which is the basis of Firefox, as well as for WebKit framework in browsers such as Google Chrome and Apple Safari [SKR+10]. These, however, require the users to download the modified web browsers in order to experience XML3D natively. Second and more recent is a *polyfill* implementation based on WebGL and JavaScript that is meant to emulate the missing parsing functionality before declarative 3D becomes widely adopted [SSK+13]. This runs in all modern WebGL-enabled web browsers including their mobile variants, see Figure 5.5. In addition, it offers functionality for external file-format loader plug-ins that can be registered for a specific MIME type, e.g. `application/json` for JSON. If multiple loaders are assigned to the same type, each is queried for support of the downloaded data block and the first in order of registration gets executed. The ability to externalise resources in XML3D further enables a form of progressive loading where downloaded components appear on the screen one by one. This returns control back to the main execution thread which prevents the client from becoming unresponsive. Such a flexible approach supports implementations of XML, JSON, BSON, SIG, OpenCTM and ArrayBuffers delivery formats, each of which sets its own challenges. These representations were evaluated in Section 5.5 for their encoding efficiency and speed of delivery.

Figure 5.5: *XML3DRepo in Mozilla Firefox web browser on tablet. Left and right arrows in the top right enable selection of previous and next revisions respectively. Model courtesy of Balfour Beatty.*

XML and JSON can utilise native parsing capabilities of modern web browsers. Binary formats, however, require development of custom loader plug-ins for their parsing. In the case of BSON, the downloaded data block is first deserialised into a JavaScript object using a client-side library [Sil12]. Vertices, faces and normals are then transformed into Xflow buffers for rendering. Similarly, OpenCTM comes with a ready-made JavaScript decoder [Gee09] that can be simply wrapped into a loader as required by XML3D. In contrast, SIG encodes 3D geometry as binary 2D images. This requires generation of an implicit vertex buffer that acts as a set of texture coordinates for the input textures. These depend heavily on the number of vertices in a mesh as well as the resolution of the images. Hence, when processing SIG data, Xflow was used to create texture coordinates so that the server delivers a mesh node that references an Xflow graph that in turn references the images. In order to minimise any unnecessary overheads, all Xflow graphs were clustered into a single external resource so that only one additional HTTP request was generated. Since the new API supports retrieval of collections of resources, single resources as well as their individual attributes, see `GET` definition in Section 5.2, it is possible to query separate images per eight-bit sections of the vertex and normal arrays as required by SIG. Finally, ArrayBuffers were implemented similarly to *BinaryGeometry* by Behr et al. [BJFS12]. With the introduction of *Typed Array* [Khr13] specification, it is now possible to load and decode vertex data on the GPU with very little processing required in JavaScript. The data arrives as an `ArrayBuffer` object from which a Typed Array view is generated and uploaded as a bound Vertex Buffer Object (VBO) to the GPU. This sets a baseline when comparing various ways of encoding 3D resources for the web.

3D Scene	Vertices	Faces	Meshes	Materials	Textures
UT4 Baeza	93,609	31,203	196	196	147
UT4 Paris v2	129,801	43,267	291	291	247
UT4 Intermodal Beta	170,717	56,915	482	482	418

Table 5.1: *Statistics for the evaluated 3D scenes. These models were selected due to their increasing complexity in terms of geometry as well as scene structure.*

5.4.2 Caching

Caching is a vital part of any such a client-server architecture. The aim is to reduce the latency and network traffic by serving locally stored copies of the requested data. For instance, it is common for web servers to cache resources that are accessed frequently. In large deployments, content delivery networks (CDNs) are used to replicate information across different physical locations that are closer to the end users in order to ensure high availability and improve performance [VP03]. Web clients cache HTML pages and external files so that only modified resources are downloaded when visiting the same URL again. In XML3DRepo system, to prevent too many connections being open from the server application to the underlying database, the connection to each repository is cached when opened and deleted via a callback when dropped. This is necessary because hundreds of requests might be needed for a single 3D scene, see Table 5.2 for examples. MongoDB itself will also cache "hot data", i.e. frequently accessed BSON documents, in memory. This raw information has to be processed into requested encoding format before it can be served to the client. Given that the 3D data in the proposed system is version controlled, the revisions will never change. Therefore, these can be cached at the application level above the database. In 3D Repo, even if the data is updated or deleted, the original revision will always be accessible via its UID. Hence, the UID together with the encoding format can act as a caching key. The same principle, however, cannot be applied to those resources that are accessed via their SID values as these often refer to the head revision and that changes over time. Sections 3.2 and 5.2 explain the difference in UID and SID use cases and interpretations. In addition, the resources can be cached directly in the web browsers as well as proxy servers on the way to the end user. HTTP provides three in-built controlling mechanisms in form of *freshness*, *validation* and *invalidation* [FGM+99]. This functionality is controlled using the response headers either as an absolute expiry or a last modification time, although, neither would apply if the connection was encrypted or the user authenticated. Still, these headers set only conditions under which the applications are not meant to keep the data. When and where a cache is emptied is purely implementation dependent and may differ from client to client.

5.5 Evaluation

Performance of the different 3D delivery formats listed in §5.1.2 was measured in a formative system study. The aim was to test the suitability of the API for 3D content delivery, and to provide indicative values for both the cumulative central processing unit (CPU) decoding time as well as the overall download time in a variety of scene sizes and web browsers. As shown in Figure 5.6, the 3D scenes used in the experiments are readily available game levels. These were chosen because they gradually increase in geometric and scene structure complexity as summarised in Table 5.1. They are also real-world exam-

Format	UT4 Baeza			UT4 Paris v2			UT4 Intermodal Beta		
	Size [MB]			Size [MB]			Size [MB]		
	Raw	Gzip	Requests	Raw	Gzip	Requests	Raw	Gzip	Requests
XML	8.9	1.4	408	7.7	1.4	598	11.9	2.4	980
JSON	8.8	1.3	408	7.6	1.4	598	11.9	2.3	980
BSON	10.4	2.8	408	10.9	3.0	598	15.7	4.6	980
SIG 8-bit	1.7	1.0	800	2.2	1.2	1,180	2.5	1.7	1,944
SIG 16-bit	2.3	1.6	1,192	2.5	1.6	1,762	3.2	2.4	2,908
SIG 24-bit	2.7	1.9	1,584	2.9	2.1	2,322	3.9	3.3	3,872
SIG 32-bit	3.3	2.5	1,976	3.6	2.9	2,926	4.8	4.0	4,836
OpenCTM	1.6	0.8	408	1.7	0.9	598	2.1	1.3	980
ArrayBuffers	3.7	1.2	1,192	4.1	2.7	1,762	5.9	4.0	2,908

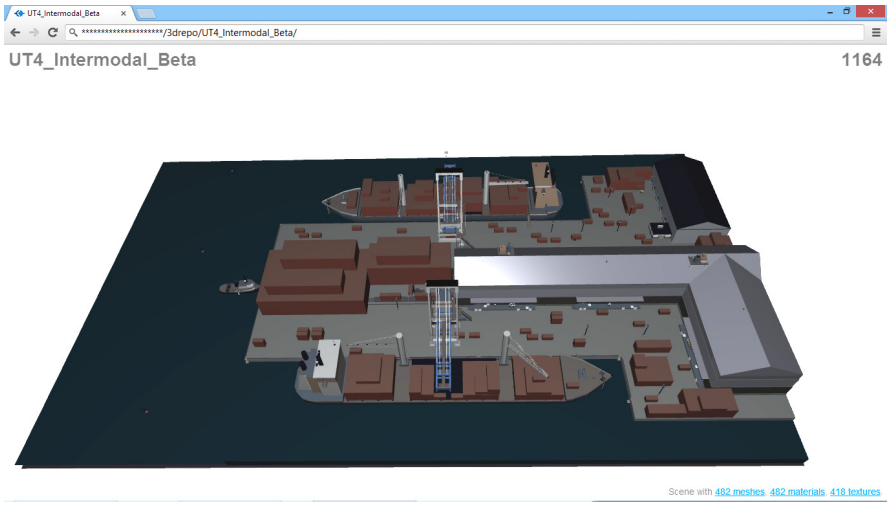
Table 5.2: *Compression and performance comparison across different representations. Overall download size of uncompressed (Raw) vs. compressed (Gzip) encodings in MB and the total number of requests for models used in the system evaluation in Section 5.5.*

ples of the types of 3D data that are commonly accessed over the Internet. Apart from materials, textures and other visual effects were excluded from the experiments to make sure the overhead for each scene would be the same regardless of the specific delivery format. These 3D models were evaluated using the built-in *Developer Tools* in Google Chrome 25.0.1364.172, *Firebug* add-on [Hew06] in Mozilla Firefox 19.02 and the built-in *Dragonfly* tool in Opera 12.14, all with caching disabled. All measurements were exported as HTTP Archive (HAR) files [Wor12]. Such files record traces of a web browser's interaction with a server in a standardised JSON notation. Apple Safari 5 and Microsoft Internet Explorer 10 were not tested as they did not support WebGL on Windows at the time of the study. An XPC Shuttle SX58H7 with Intel Core i7-920 at 2.67 GHz with 4 GB RAM running Microsoft Windows 7 was used as a desktop client. On the server-side, Intel Xeon at 2.67 GHz with 2 GB RAM running Community Enterprise Operating System (CentOS) 6.2 and node.js 0.8.19 acted as a remote host over a local network with an average round trip time of 6 ms. In order to mitigate the effects of data handling between the application server and DB, all experiments were performed with MongoDB's C++ BSON parser/serialiser enabled.

Despite using only eight-bits per normal array in the prototype implementation in Section 5.4, SIG 32-bit still requires four textures for the vertex array definition alone. This means that the resulting 3D models are rendered without textures or shading even though the data has been downloaded. Albeit not visually acceptable, measurements from these models are still included for completeness. Table 5.2 shows the different compression rates achieved by individual encoding formats as well as the number of requests made by the web browsers when retrieving 3D scenes from the server. Figure 5.7 charts median values from uncompressed encodings measured across five trials. This is to suppress fluctuations in the network and CPU performance when evaluating the system. The overall download times are composed of the time required to load the DOM definition as well as the external resources. In addition, the yellow dots show the cumulative CPU decoding overhead in JavaScript per format. Unfortunately, the SIG format did not work in Opera. When compiling a vertex shader with texture fetches it failed due to a DirectX-specific error despite reporting available texture units. Results from compressed measurements are similar due to the size of the data and the decompression overheads.

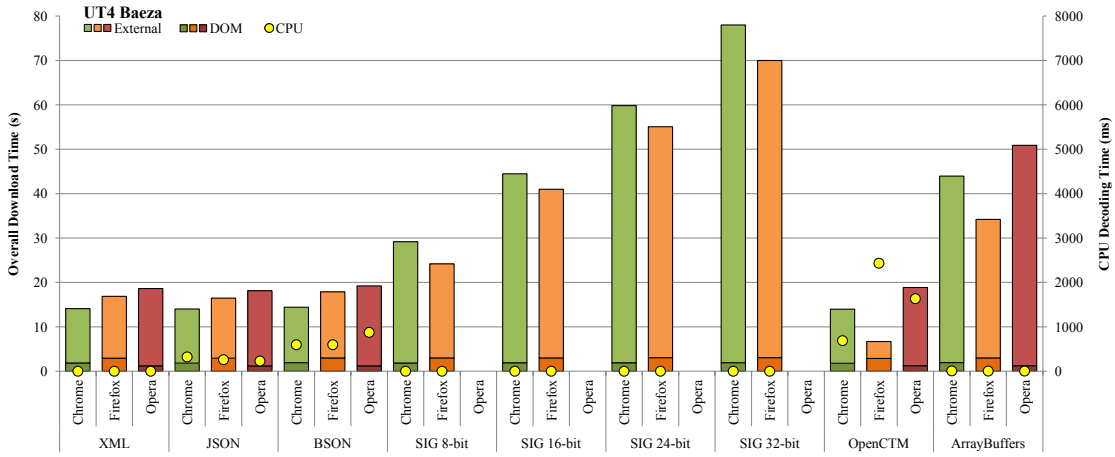
(a) *UT4 Baeza*

(b) *UT4 Paris v2*

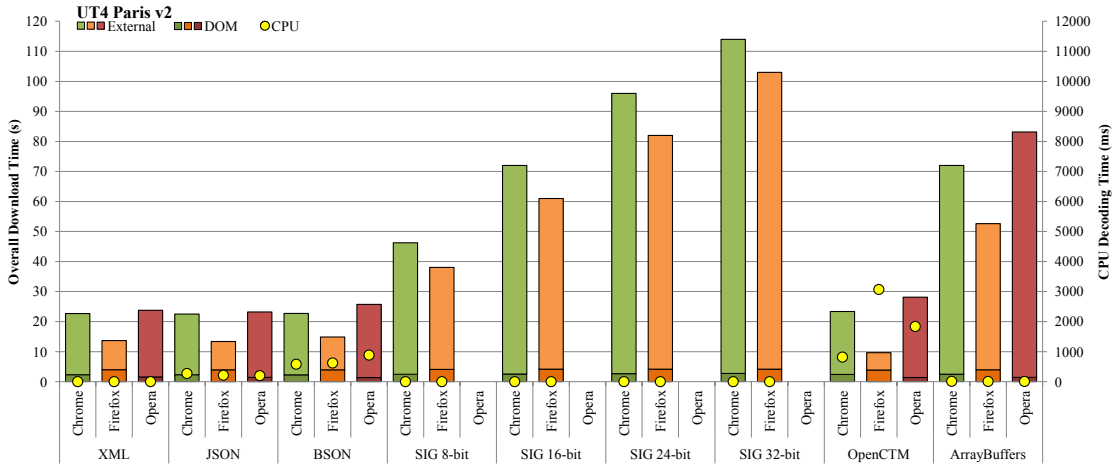


(c) *UT4 Intermodal Beta*

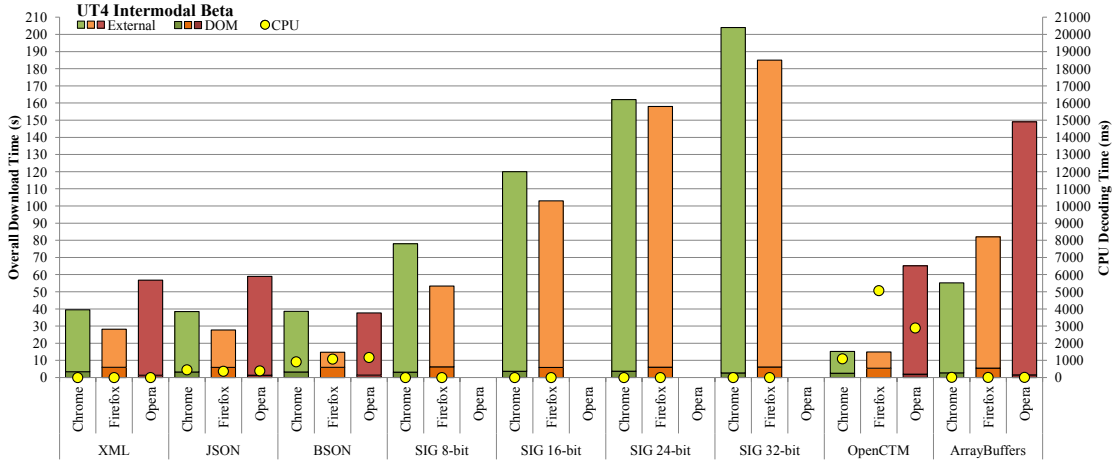
Figure 5.6: *Three game levels used in XML3DRepo experiments. From top to bottom: UT4 Baeza with over 93,000 vertices, UT4 Paris v2 with over 129,000 vertices and UT4 Intermodal Beta with over 170,000 vertices. Textures and other visual effects were deliberately excluded not to interfere with the measurements. Models courtesy of Sniper Gaulois.*



(a) *UT4 Baeza*



(b) *UT4 Paris v2*



(c) *UT4 Intermodal Beta*

Figure 5.7: *Median values from five trials of three game levels. Overall download time (left Y-axis) consists of a DOM definition and the external references processing while the CPU time (ten folds less, right Y-axis) defines the amount of cumulative CPU milliseconds required to decode the format. Measured on XPC Shuttle SX58H7 with Intel Core i7-920 CPU at 2.67 GHz with 4 GB RAM running Windows 7.*

5.6 Discussion

In the experiments in Section 5.5, all resources were queried independently. The additional 16 requests constitute a constant overhead in form of a scaffolding DOM definition, individual loaders required by XML3D and supporting JavaScript libraries that make up the web client infrastructure. Since these are the same regardless of the scene being loaded, they do not contribute to the differences in Figure 5.7. In a production environment, these supporting files would be minified, i.e. all redundant information such as white spaces, comments and new line characters would be automatically removed. Also, only the specific loaders required for each resource type would be included, all of which are likely to be cached by the browsers anyway. It is even customary to group multiple JavaScript files into a single larger library in order to reduce the number of XMLHttpRequests (XHRs) from the client to the server.

As expected, the differences in data sizes between XML and JSON in Table 5.2 are very small due to the 3D scenes consisting mostly of large vertex, normal and face arrays. Therefore, the contribution of the end tags in XML is diminished when comparing to JSON. In addition, XML can group multiple resources into a single file where each can be accessed directly using URI semantics, unlike JSON which can only represent single resources. Figure 5.2 shows an example. By way of contrast, BSON representation is noticeably larger than either JSON or XML despite being binary. This is because of the explicit array indices that cause a massive overhead in 3D data. For this reason alone, 3D Repo relies on unstructured binary entries for its internal representation of meshes. Unfortunately, unlike ArrayBuffers, such binary blobs cannot be uploaded directly to the GPU since BSON documents have additional header information that has to be removed before rendering. Nevertheless, even with a larger size and a deserialisation requirement, BSON performed similarly to both XML and JSON. The number of requests was the same and the overall download time was very comparable, too. The only major difference is the increase in JavaScript CPU processing that would disadvantage BSON on less powerful clients such as mobiles or tablets. In contrast, SIG decodes in native code and therefore achieves zero CPU overhead. However, the number of HTTP requests increases with the increasing precision and this significantly hampers the overall download time despite a much smaller amount of data being transmitted. What is more, the lower-bit quantisations cause visual degradation especially in those areas where there are high frequency details. In architectural models such as those shown in Figures 5.4 and 5.5, the lower floating point precision causes cracks between walls to appear on flat surfaces. On the other hand, OpenCTM offers lossless compression that does not suffer from any visual artefacts. The advantage of this encoding format becomes apparent with increasing model sizes. Nevertheless, Firefox would consume more CPU to decode this format which must be specific to its internal JavaScript engine implementation. On the other hand, Opera would take much longer to download and display the entire 3D scene in spite of lower CPU overheads. Note that the evaluated version of Opera uses a proprietary Presto [Ope03] layout engine which as of version 15 was replaced by Blink [Goo13], a fork of WebKit [App98], that is also used in Google Chrome. Therefore, in the later releases of Opera the results should be similar to those of Chrome web browser. Finally, ArrayBuffers provide baseline results in that there is no CPU overhead and they encode each 3D buffer independently for a direct GPU upload.

Based on these results it is concluded that the best format can be only selected by considering the properties of the network connection and the decoding performance. The former is becoming less of a bottleneck nowadays due to the ever increasing processing capabilities of clients, nearly all of which support WebGL already. This system can be even executed on memory limited devices such as mobiles or tablets as shown in Figure 5.5. The network connection is, therefore, the key factor for selecting the most suitable delivery format. High latency requires fewer requests while low bandwidth requires more compression. With the new API, the desired delivery format can be dynamically chosen by the client making this a very flexible system. In addition, it enables management of version controlled 3D assets in a manner that does not expose the underlying database to the end user. Thus, it enhances the 3D Repo system with the ability to enforce versioning principles even from third-party client connections.

Concurrent work to the evaluation presented in Section 5.5 is that of Limper et al. [LWS+13] that was published in the same year and at the same venue as these findings. They evaluated a text-based X3D format together with binary formats such as BinaryGeometry (ArrayBuffers) and OpenCTM. The main difference is their use of polygonal 3D models that were at least an order of magnitude smaller in their complexity. These were served as simple files rather than a larger set of decomposed resources. Nevertheless, similarly to the results discovered here, they found that a text-based encoding generates the largest data representations, although, after compression it is comparable to the tested binary formats, see our Table 5.2. They also experienced the same trade-offs between decompression time and the overall download time which are highly dependent on the available bandwidth as well as the model size and the processing power of the receiving client. Thus the two studies complement each other.

5.6.1 Limitations

A common approach when loading 3D data over the Internet is to show a progress bar with a percentage value indicating the remaining amount of data to be downloaded. This means that the user has to wait until the entire scene becomes available before any interaction begins. In cases where large engineering 3D models such as the one depicted in Figure 5.5 are being fetched, it would be certainly a long time to wait. This is because such models are not optimised in any way and often contain redundant geometry and duplicated vertices. Although a form of progressive loading has been achieved in the prototype implementation in Section 5.4 by rendering scene components as soon as they become available, it is still not a truly progressive visualisation. Representations such as progressive meshes [Hop96] enable morphing of the topology at the vertex level. The very latest POP buffers [LJBA13], for instance, support dynamic clustering and quantisation of geometry. However, such techniques are not yet supported by this system. Even with progressive morphing, the client would still require the entire scene graph structure to be available before meshes can be displayed unless components are already in their global coordinates. For visualisations on the scale of an entire city, such an approach would not be feasible as the scene structure alone might be too large to transmit even though only a portion of the entire model would be visible at any given point. What is more, in experiments with massive 3D models such as a section of the M25 motorway or the London Olympic Stadium Transformation (OST) by Balfour Beatty, it was discovered that although 3D Repo can store and visualise these models without any optimisation, the web

clients cannot. A motorway scene, for example, has over 74,000 separate components and JavaScript engines in web browsers are simply not capable of handling so many scene graph nodes. Only when the components are joined in order to reduce the scene complexity, it is possible to visualise such engineering models in web browsers. Unfortunately, the client would not be able to select individual components.

Furthermore, the proposed REST API does not make any assumptions in regards to the user privileges or authentication. In a real-world deployment, it will be necessary to restrict access to 3D models only to authorised users who can be validated via the API directly. It is common to provide means of listing user details via calls such as `/xml3drepo/users` or to log in by providing user credentials via `/xml3drepo/login`. Although, this functionality is not supported by the system, it can be easily added as an extension in the future. In addition, the choice of UUIDs for the API makes the system directly compatible with 3D Repo database schema as well as readily available for distributed access. In the future, users could run a local instance of the server application to record offline modifications and synchronise to a centralised repository where, just like in Git, the integration would happen remotely. This proposition, although conceivable, was not evaluated as it is outside the scope of this thesis.

5.7 Chapter Summary

The second research question in Chapter 1 asked whether a domain-specific VCS can deliver real-time visualisations of large scale 3D models that would be independent of their underlying data representations. Hence, a novel fusion of XML3D and 3D Repo was presented in this chapter. On one hand, XML3D is an open source extension to HTML5 that supports interactive 3D graphics in WebGL-enabled web browsers. On the other hand, 3D Repo, introduced in Chapter 3, is a domain-specific version control system for management and visualisation of non-linear history of polygonal 3D models. The initial framework, however, provides only raw access to its NoSQL database. Thus, the system proposed in this chapter defines a novel server-side daemon service which stores unified file format-independent representations of 3D scenes in 3D Repo and exposes a RESTful API for deeper integration with other services. This can deliver 3D resources in a variety of encodings that can be opted for by the client application depending on the network properties and its own processing capabilities.

The overall system architecture was outlined in Section 5.1 and a simple yet powerful API was defined in Section 5.2. The API was developed into a web system in Section 5.4 such that the client requests hundreds of resources from the server directly. Such an approach is believed to be able to accommodate crowdsourcing of 3D models in the future as the users will be able to collaborate on large scenes remotely without the fear of irrevocably overwriting previous revisions. For this purpose alone, different strategies of 3D data encoding were developed and evaluated in terms of their speed and efficiency across several popular web browsers. As the results of the study imply, the API provides a consistent online addressing of version controlled 3D assets. The main advantage is that no changes in the repository implementation were necessary because 3D Repo stores scene graph structures in BSON format independent of any XML3D formatting. Nevertheless, the novel client-server architecture demonstrates that the resource-based approach supports delivery of requested assets in a variety of encodings. In Section 5.3, it was discussed how XML3D, due to its consistent approach to external resources, would be suitable for these

purposes and how it could offer transparent use of different data representations on the web. This claim was supported by six considerably different data delivery formats in Section 5.5. The superiority of the API over a simple web server is in its ability to query entire scenes, individual components and even their attributes all in representations that are independent of the underlying data store.

Unfortunately, the experimental results show that currently there is no single delivery format that would be suitable to all applications and use cases. Even though OpenCTM offers considerable size reduction, it causes slow decompression on the client-side. This schema is applicable to triangular meshes only, so other more generic formats such as XML and JSON have to be used for other types of 3D assets. These, however, require parsing and offer only generic compression in HTTP. In contrast, the direct use of SIG or ArrayBuffers suffers from increase in the number of XHR calls. What is more, SIG is limited by the number of applicable textures on the rendering device. Therefore, it can be safely concluded that none of the tested formats provides the right balance of the number of requests, decoding overheads and compression rates. Thus, an ideal format would need to be efficient in its encoding so that it is compact for transmission but also based on TypedArrays so that it can be uploaded to the GPU directly. Furthermore, the data representation has to be structured in order to interleave multiple buffers into a single streamable request. Finally, the format should not impose any specific schema but rather provide extensibility for future data representations that might not even exist at present. Following these suggestions, two experimental proposals for new data formats based on XML3D and X3DOM respectively have been very recently put forward by Sutter et al. [SSS14] and Limper et al. [LTBF14].

Nevertheless, with the restrictions of existing data formats in place, the delivery framework proposed in this chapter becomes even more important as it offers the right means of adapting resources to the specific application needs of the receiving clients. Nevertheless, even with systems such as 3D Repo in combination with the new REST API, there is still no suitable way of examining legacy data sets that were not preserved using such a version control framework in the first place. Chapter 6, therefore, introduces a novel system for reverse engineering of modelling history from independent 3D files. This provides an interactive visualisation interface in the form of a 3D Timeline which displays dependencies across components as well as editing operations that have been performed between revisions.

Chapter 6

3D Timeline Reverse Engineering

A trend that has been identified in Chapter 1 is the need for organisation and inspection of large collections of polygonal 3D models. These range from engineering and architectural visualisations to games development and even 3D printing applications. Chapter 3 introduced a unique versioning system that is capable of preserving deconstructed scene components in a NoSQL database. This framework sets the granularity of detected changes at the scene node level which is appropriate in most situations where models encompass thousands of disparate 3D assets. The associated 3D Diff tool from Chapter 4 further performs 2-way and 3-way differencing and merging so that concurrent changes can be combined into a single coherent result. Finally, a novel client-server architecture developed in Chapter 5 enables retrieval and management of version controlled 3D assets in a variety of encoding formats over the Internet.

Nevertheless, the outstanding research question in Chapter 1 asks whether it is possible to imply detailed editing history from legacy datasets especially if these were not stored in a VCS. This leads to what might be called *"a disk full of models problem"*: there is a disk, or a version control repository, with various revisions of 3D models, but understanding the provenance of these potentially large datasets is no easy task due to very little associated metadata being available. This chapter, therefore, presents a novel tool for reverse engineering of modelling history from 3D files based on a timeline abstraction. Although the timeline interface is commonly used in 3D modelling packages for animation compositing, it is much less frequently used in geometry editing visualisation. Unlike previous methods listed in §2.3.2 that require instrumentation of the editing software, or even 3D Repo which records delta changes on every commit, this new approach does not rely on pre-recorded editing instructions. Instead, each stand-alone 3D file is treated as a keyframe of a construction flow from which the editing provenance is reverse engineered. Hence, the concept of a sequential differencing introduced in Chapter 4 is utilised in a novel algorithm. In Section 6.2, the input models are segmented into separate components and their correspondence is estimated. This is then analysed for high-level semantic operations in Section 6.3 and the results are visualised as an interactive timeline in Section 6.4. The algorithm was evaluated on six complex 3D sequences created in a variety of modelling tools by different professional artists. A comparative user study in Section 6.5 concludes that such a visualisation is well suited to the purposes of provenance extraction and inspection of large datasets. Thus the contribution is in providing a novel reverse engineering solution for extraction of semantic operations when no revision metadata is available.

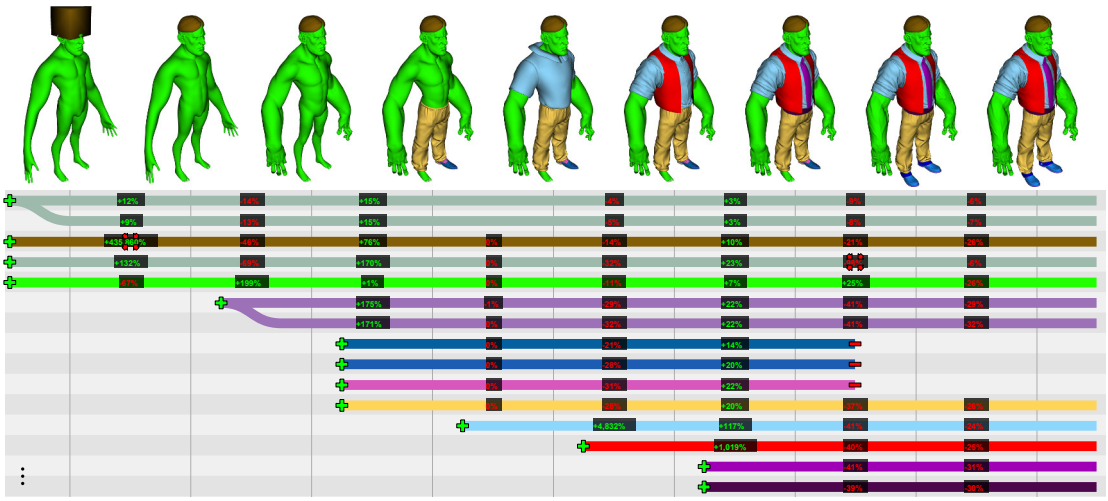


Figure 6.1: *Extracted and collapsed editing timeline (bottom) from 9 keyframes (top) of a modelling sequence with over 7.6 million polygons in total. Legend to the detected operations is listed in Figure 6.4.*

6.1 System Overview

A frequently examined problem is the categorisation of large collections of 3D scenes, see §2.3.3. Such collections might come from archives of similar models that need to be classified by type or shape. This chapter, however, addresses a complementary problem of the organisation of polygonal 3D models in a temporal domain of editing history. This problem arises since many authoring tools do not save the editing operations, and even if they do, only for a few recent steps. Native histories can anyway be manually deleted or lost when exporting into interchange file formats. Although most tools allow file names to be auto-increment and auto-saved, management of such files is poorly supported. Even if the 3D models were version controlled in a system such as 3D Repo, the changes would be recorded only between immediate revisions without any further semantic explanation of the steps that guided the creation of the entire data sequence. This is often experienced in the industries where projects span multiple specialised engineering tools. A common practice is to periodically save snapshots of whole 3D files for archival purposes. Thus, a novel tool that takes a set of models and builds a visualisation of a high-level editing provenance is presented here. The tool does this by reverse engineering a *plausible* history of 3D components and then summarising the important changes. These are displayed on a timeline that makes it easy to visually track the lifetime of each part and its relationship to other parts. Using this tool, the users are able to answer important questions about the model history, such as the timing of a particular change or the steps that introduced an error. Figure 6.1 shows an example.

The Chronicle system by Grossman et al. [GMF10] visualises the history of documents in a timeline, but to do so, it has to instrument an editor and video record the entire session. This system has been later extended to Autodesk's computer-aided design (CAD) software, too, although it is still not applicable to legacy datasets where such a video capture might not exist. This is certainly the case for the majority of already created 3D models that one might want to inspect. Hence, a newly developed *3D Timeline* tool has to be totally agnostic to the editors that generated the models and has to take as input only complete 3D files. In comparison to Chronicle, 3D Timeline does not require any instrumentation of

the editor, nor any pre-recorded editing sequences. To demonstrate the feasibility of the proposed solution, models saved from Autodesk Maya [Pal13], Pixologic ZBrush [Spe11], Trimble Sketchup [Sch13], Blender [Bla12] and Luxology Modo [Gar13] were used in the evaluation in Section 6.5. These sequences that are tackled initially consist of millions of polygons and thousands of components, see Table 6.1 for statistics. In order to make it tractable from an analytical point of view, and to achieve a concise visualisation, this work is based on the observation that many models are composed of separate parts rather than single manifold surfaces. These parts might reflect logical structure, or might simply be a facet of the working process of the modeller. In addition, models are often comprised of duplicated, symmetric or self-similar parts [MPWC13]. Hence, the focus is on reverse engineering of changes in parts and the aspects of duplication, provenance from common roots and instantiation. Similar to the very recent inverse image editing [HXM+13] is the set of simple geometric rules and methods proposed in §6.3.1. The aim is to detect common editing operations and extract their semantic provenance. In addition, 3D Timeline provides a compression by collapsing non-conflicting edits and removing redundant models. The analysed and abstracted timeline can then be displayed in a custom viewer in Section 6.4 to quickly browse over the history and key events. The system was tested on a selection of models, each with thousands of components. This successfully recovered informative summaries, see Figure 6.9.

6.1.1 System architecture

Given an input set of consecutive polygonal 3D models, also known as *keyframes*, the goal is to reverse engineer a modelling tree that explains their temporal relationships. In order to extract high-level semantic meaning, the changes need to be explained beyond simple version control operations. This is achieved by decomposing the keyframes into their constituent parts, and for each part, by tracking its *provenance*, i.e. how it is related to identical or similar parts in adjacent frames. The full editing timeline will then explain the life-span of every identified model component. Since this is an inverse problem, the task is inherently ambiguous. This is because several intermediate edits can potentially explain the same input model sequence. Hence, 3D Timeline does not attempt to recover the actual history, nor does it attempt to exhaustively generate all possible permutations of the histories. Instead, it tries to infer a plausible flow of steps that fulfil the assumptions with regards to the permitted editing operations. This flow has to provide a consistent explanation of the geometry found in the input models. As shown in Figure 6.1, the detected operations together with their visualisation and playback form the core output of the system. Even though the creative process of modelling might be regarded as continuous in the temporal domain, here, it is discretised into individual events that are defined in §6.3.1. These include the following basic component-level operations: *addition* and *deletion*, changes in *polycount* and *size* of the corresponding parts as well as detection of *translations*, *duplications*, *instancing*, and *repeated copying*. Such operations are dominant in edit histories as validated on various modelling workflows in §6.5. Thus understanding them provides valuable insight, despite not necessarily capturing all of the steps an artist might have performed. Even though the core 3D Repo system already tracks changes as *additions*, *deletions* and *modifications*, the crucial difference is that 3D Timeline does not rely on any prior knowledge of revision metadata. Instead, it automatically extracts and estimates a component-level correspondence

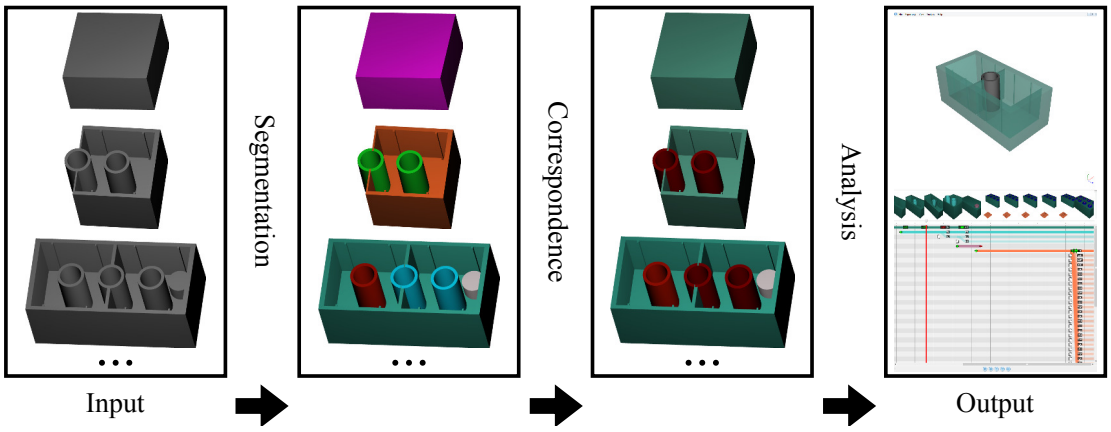


Figure 6.2: *3D Timeline processing pipeline. A collection of consecutive models is loaded. In a pre-processing step the meshes are individually segmented and a part-based correspondence is estimated. This is analysed and the implied editing history is visualised.*

flow across the input frames. This provides the right balance of complexity vs. usefulness. Subsequently, the system implies the editing operations and visualises them using a timeline metaphor. The users can then scrub through the timeline, as they might in a video editor, and an animation is created that demonstrates how the model evolved over time. Similarly to Grossman et al. [GMF10], a timeline compression simplifies the displayed information without violating the provenance, see §6.3.3. In rare cases when the automatic correspondence fails, the users can override the assignment.

6.1.2 Processing pipeline

The input keyframes, or modelling "snapshots" as they can be referred to, provide direct evidence from which a reverse engineering solution implies the missing editing steps that were not recorded, see Figure 6.2 for summary. The system, therefore, proceeds in the following stages:

1. The input of the algorithm is a collection of 3D files composed of polygonal meshes that represent a linear evolution of a scene. Their temporal ordering is assumed to be known as it is often explicitly recorded using naming conventions when modelling. Alternatively, such a sequence can be exported from a VCS such as 3D Repo, see §6.6.1 for limitations.

2. In the pre-processing step in Section 6.2, the system performs an independent component analysis in order to extract model segments and to establish their mutual correspondence across all input frames. This is a multi-stage process in itself. First, self-similar groups of the components are identified and the correspondence is propagated backwards from the last to the very first frame.

3. Semantic analysis in Section 6.3 detects editing operations within estimated correspondence flows. The extracted operations are then grouped, and possibly collapsed by merging non-conflicting edits over time in order to reduce the inherent complexity of the detected editing.

4. Finally, the implied provenance is visualised as a timeline in Section 6.4 with colour-coded correspondence, highlighted events, and a playback option. The compressed timelines provide succinct edit summaries showing a quick overview particularly in long sequences.

6.2 Pre-processing

Artists typically create, represent and manipulate shapes as collections of components that are commonly supported by various primitive-based modelling tools. In contrast to co-analysis of model collections, reverse engineering from editing sessions of 3D models poses different problems altogether. Generally speaking, when assigning correspondence in any two 3D scenes, those vertices and faces that are identical in world coordinates might not necessarily correspond to the same part of the overall geometry due to often localised discrepancies, e.g. a subdivided plane being shifted half its width sideways. As demonstrated in [GF09], relying strictly on pre-alignment of the meshes can yield poor results. Therefore, 3D Timeline makes no assumptions with regards to the initial global alignment of the models. Instead, it builds upon the observation that the early design stages are often characterised by *massing*, i.e. outlining of the most prominent volumes first before progressively adding detail later [ES11]. The focus is on establishing a component-level correspondence across frames, starting with the largest dominant components that then provide additional contextual information for the less prominent parts. This factors out any global rigid transforms. Furthermore, the neighbouring models in a construction sequence tend to be highly correlated with many sections being locally unmodified. 3D Timeline takes advantage of this characteristic to formulate a correspondence estimation in §6.2.2.

6.2.1 Segmentation

Many segmentation strategies exist, e.g. *randomised cuts* [GF08], *joint shape segmentation* [HKG11] or *unsupervised co-segmentation* [SvKK+11, WWS+13, MXLH13] to name just a few. These, however, are computationally expensive and rely on specific descriptors and a fixed number of oversegmentation clusters. In contrast, 3D Timeline requires fast segmentation for pre-processing only. Thus, the classical *hierarchical face clustering* by Garland et al. [GWH01] is used to independently generate non-overlapping components as clusters. This algorithm builds a dual graph of a mesh surface such that nodes represent clusters initially seeded by individual faces. Edges, ordered according to their cost of collapsing, represent the adjacency of faces in this graph. At each iteration, the lowest cost edge is removed, its clusters are merged and the costs of their inherited edges are recalculated. The best-fitting plane for a collection V of vertices $v_i \in V$ is defined by their mean vertex $\bar{v} = \frac{1}{|V|}\sum_i v_i$, and a normal n of the plane which is the eigenvector corresponding to the minimum eigenvalue of their covariance matrix $Cov_v = \sum_i (v_i - \bar{v})(v_i - \bar{v})^T$. The L^2 fitting error to this plane is then defined as $L^2 = \sum_i [n(v_i - \bar{v})]^2$. In addition, the *compact shape bias* adds a penalty based on the relative change in the clusters' irregularity in order to prevent long skinny and otherwise degenerate clusters. This irregularity is defined as $\rho^2/(4\pi A)$, where ρ is the perimeter and A the area of a cluster which is simply initialised by the polygon faces and grows as the sum of the collapsed areas. However, the perimeter of a cluster is the sum of its two previous perimeters minus twice the length of their common boundary. This boundary length is stored in each edge. When collapsing two clusters, the new boundary lengths are the sums of the boundaries with the neighbours they had in common. If the algorithm proceeds until all edges have been exhausted, it effectively identifies disjoint manifold components. Apart from clusters' planarity, the cost can express conformity to shapes such as spheres and cylinders, see Attene et al. [AFS06].

6.2.2 Correspondence flow estimation

In order to establish a correspondence flow

$$F = \{C_{t_i} \to C_{t_{i+1}} \to \ldots \to C_{t_{i+n}}\}, \tag{6.1}$$

i.e. an assignment of a component C at time t to a component C' at time t', it is simply not sufficient to find the most similar meshes since components can be deformed, refined, copied, etc. Instead, the exact same component needs to be identified and tracked across all keyframes so that its provenance can be reliably determined. The correspondence measure has to be robust to changes in shape and location, yet it has to discriminate duplication. Therefore, four consecutive steps are executed as follows:

PCA-aligned bounding boxes. Although complex algorithms for calculating the exact as well as approximate minimal-volume bounding boxes in 3D exist [O'R85, BHP99], in the interest of tractability, a strategy similar to that of Jain et al. [JTRS12] is used here. To begin with, a principal component analysis (PCA)-aligned bounding box is calculated in order to establish a rough descriptor for each component. PCA of vertices v_i in a component C weighted by the cumulative area A_i of the parent faces provides a transformation from the global to a local coordinate system. Let $C_\square[\bar{v}, w, h, d]$ be the bounding box of C in this coordinate space with a centroid $\bar{v} = \sum_{i=1}^{n} A_i v_i / \sum_{i=1}^{n} A_i$, and $[w, h, d]$ its respective width, height and depth. This bounding box is easy to compute, reflects the spread of vertices along the principal axes, is rotation invariant and robust to local geometry modifications.

Part-based hierarchy. Building a part-based hierarchy is a common way of adding contextual support and relative localisation to otherwise loose components [SSS+10, JTRS12]. Instead of creating a complicated tree of components, in this step at each keyframe independently, a *forest* of one-level deep trees rooted at the largest components is defined. Hence, the parent C_P of a component C is the one that contains its bounding box and has the largest volume. This provides localised systems where the largest components have an implicit global reference at the origin. Note that the parent might not be the same part of a scene as components can be translated, disconnected, etc.

Correspondence estimation. Based on the bounding boxes and their hierarchy, correspondence between components can be estimated. Let E_S be a similarity error between two components C and C' defined as a Euclidean distance of their bounding boxes irrespective of their centroids $\bar{v}, \bar{v}'$

$$E_S := \|C_\square[w, h, d] - C'_\square[w', h', d']\|. \tag{6.2}$$

This measure is used to group self-similar components in each keyframe, step (1) in Figure 6.3. Let E_L be a localisation error defined as an absolute difference of the Euclidean distances of the centroids of the component bounding boxes $C_\square, C'_\square$ to the centroids of the bounding boxes of their associated parental components $C_{P_\square}, C'_{P'_\square}$ as

$$E_L := \left| \|\bar{v} - \bar{v}_P\| - \|\bar{v}' - \bar{v}'_{P'}\| \right|. \tag{6.3}$$

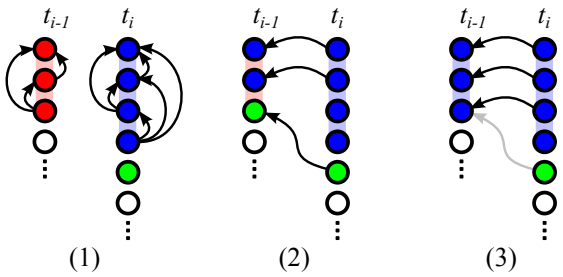


Figure 6.3: *Correspondence assignment from time t_i to t_{i-1}. (1) Firstly, independent self-similar groups are established in each keyframe. (2) Next, a one to one correspondence is assigned. (3) Finally, a majority vote ensures all correspondences in t_{i-1} come from a single group in t_i.*

Combining Equations 6.2 and 6.3 for each pair of components in two frames leads to an affinity matrix S as a weighted sum of the similarity and localisation errors:

$$S_{i,j} := \alpha E_S + (1-\alpha) E_L. \tag{6.4}$$

Correspondence propagation. Finally, a greedy one to one assignment based on the affinity matrix S estimates the initial correspondence from frame t_i to t_{i-1}, see step (2) in Figure 6.3. This assignment is then evaluated for consistency based on a majority vote within each group. Outliers at t_{i-1} with correspondence assigned from a non-matching self-similar group at t_i are reassigned to the remaining components of the group at t_i that was voted as a majority preference. This step enforces a consistent flow between groups of components rather than individuals, see step (3) in Figure 6.3. The system proceeds through pairs of neighbouring frames in this fashion from the last to the first. If a correspondence cannot be reliably established between two frames, the frame is skipped in order to seek a match in the next frame until a suitable candidate, if any, is found. The self-similar groups ensure that the components gain not only a one to one but also a one to many correspondences, see the "funnels" in Figure 6.9.

6.3 Semantic Analysis

As illustrated in Equation 6.5, the extracted correspondence flows from §6.2.2 can be represented as a sparse binary $m \times n$ matrix Φ, where m is the number of flows and n the number of keyframes.

$$\Phi_{m,n} = \begin{array}{c} \\ F_1 \\ F_2 \\ \vdots \\ F_m \end{array} \begin{array}{c} \begin{array}{cccc} t_1 & t_2 & \cdots & t_n \end{array} \\ \begin{pmatrix} C_{1,1} & C_{1,2} & \cdots & C_{1,n} \\ C_{2,1} & C_{2,2} & \cdots & C_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ C_{m,1} & C_{m,2} & \cdots & C_{m,n} \end{pmatrix} \end{array} \tag{6.5}$$

Entries in Φ express the presence of a component $C_{i,j}$ in keyframe at time t_j. Hence, each row defines a single logical scene part tracked over time, while columns define collections of components that belong to a particular keyframe. In this representation, the natural temporal ordering is from left to right, i.e. from the first to the last keyframe. By sorting the rows such that a presence in an earlier column t and more overall entries across all columns are favoured, a top to bottom temporal ordering is created.

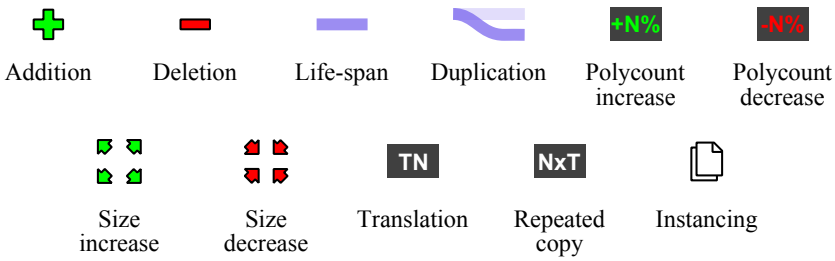


Figure 6.4: *3D Timeline legend. Each icon signifies a detected operation in the resulting timeline.*

6.3.1 Editing operations

Once the correspondence flows have been organised into the matrix Φ, the part by part changes between the pairs of keyframes can be examined. These are classified into one or more of the following operations such that each operation is assigned a visual icon as depicted in Figure 6.4.

Addition. A component $C_{i,j}$ has been *added* between t_{j-1} and t_j iff it is the first in its self-similar group and there is no associated corresponding component $C_{i,j-1}$.

Deletion. Conversely, a component $C_{i,j}$ has been *deleted* between t_j and t_{j+1} iff no correspondence at t_{j+1} exists. However, components that are present in the last keyframe provide no more evidence to support the deletion detection, hence are left unlabelled.

Life-span. The time distance between a component being added and deleted represents its *life-span*.

Duplication. A component $C_{i,j}$ is a *duplicate* iff it is added but not a *template*, i.e. not the first in its group. The choice of a template is implementation dependent but, in general, can be arbitrary.

Polycount increase. A component $C_{i,j}$ has *increased in polycount* between t_{j-1} and t_j iff it has a larger number of polygons than $C_{i,j-1}$.

Polycount decrease. Conversely, a component $C_{i,j}$ has *decreased in polycount* between t_{j-1} and t_j iff it has a smaller number of polygons than $C_{i,j-1}$.

Size increase. A component $C_{i,j}$ has *increased in size* between t_{j-1} and t_j iff its bounding box volume is larger than that of $C_{i,j-1}$.

Size decrease. Conversely, a component $C_{i,j}$ has *decreased in size* between t_{j-1} and t_j iff its bounding box volume is smaller than that of $C_{i,j-1}$.

Translation. A component $C_{i,j}$ has been *translated* between t_{j-1} and t_j iff there is a translation T signifying a difference in the global position of its bounding box centroid $\bar{v}_{i,j}$ to that of $C_{i,j-1}$ or to its template at t_j if it is a duplicate.

Repeated copy. A component $C_{i,j}$ is a *repeated copy* between t_{j-1} and t_j iff it is a duplicate and its T belongs to a list of at least three successive translations, see §6.3.2.

Instancing. A component $C_{i,j}$ is *instanced* iff it is a duplicate and its life-span operations match all of those present in its template.

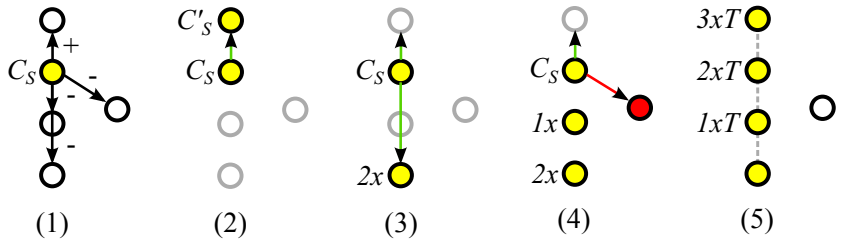


Figure 6.5: *Repeated copying detection. (1) Distances from an arbitrary seed C_S are calculated. (2) The smallest vector defines the desired line direction. (3) Components with parallel vectors are selected and unparallel (4) rejected. (5) The furthest component becomes the new template.*

This semantic labelling iterates through each row of matrix Φ and for each column it detects such operations through a lookup table. Here, it is important to process the data row by row since instancing needs to compare all operations of the template component with the currently examined one. Certain operations such as changes in polycount and size or instancing and repeated copying can and often do occur simultaneously. This needs to be taken into account when visualising the timeline, see Section 6.4. Additional operations such as rotation and scaling were not considered. This is because PCA-aligned bounding boxes are rotation invariant and their scaling is ambiguous unless the component-level correspondence is known beforehand.

6.3.2 Repeated copying detection

Apart from instancing, another special case of duplication is repeated copying. A template component that was duplicated and belongs to a self-similar group G_i at time t_i is a component with the largest lifetime. In a case of multiple candidates fulfilling this criterion, the choice would be arbitrary. Repeated copying, however, differs from basic duplication in that the translation from the template to each copy is repetitive, i.e. it can be expressed as an incremental succession of the same translation T such that the most immediate copy is assigned $1 \times T$, the next $2 \times T$ and so on, while T is minimal. Essentially, the repeated copying detection is looking for a 1-parameter regular structure, c.f. [PMW+08], where the component-based instances and repetitions are exact, thus easier to discover. The focus here is on duplicates that are equally spaced.

Given a self-similar group G, an arbitrary component $C_S \in G$ is selected as a seed. The aim is to identify at least two other components that would form a repeated sequence with C_S, their spatial ordering, the template component and the translation T that governs this repetition. In order to unravel such repeated copying, the detection algorithm proceeds as follows.

Firstly, distances from C_S to all other components are calculated, see step (1) in Figure 6.5. Let $\overrightarrow{AC_S}$ be a vector from any arbitrary component $A \in G \wedge A \neq C_S$. A distance from C_S to $B \in G$ is considered *negative* iff $\overrightarrow{AC_S} \cdot \overrightarrow{BC_S} < 0$. The components are, therefore, sorted according to their signed distance from the most negative to the most positive with C_S also being included due to its trivial non-negative zero distance to itself. Let $C'_S \neq C_S$ be the component with the smallest unsigned distance, in absolute terms, to C_S. Vector $\overrightarrow{C_S C'_S}$ then defines a line on which the repeated copying is expected to occur while its magnitude defines the desired repetition distance, see step (2) in Figure 6.5. Next, the components

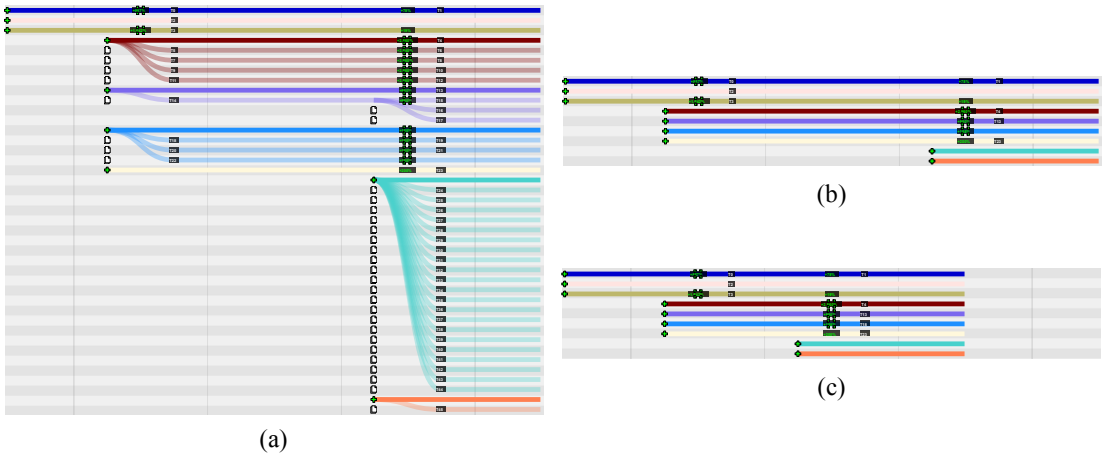


Figure 6.6: *Timeline compression. (a) Full timeline. (b) Row-wise collapse of instanced duplication. (c) Column-wise collapse of (b) where no colliding operations were detected.*

are examined one by one in the order of their signed distances. Component D is considered a member of a repeated copying subgroup iff $\|\overrightarrow{C_SD} \times \overrightarrow{C_SC'_S}\| = 0$, i.e. the vector from a seed component to D is parallel with the desired line and $mod(\|\overrightarrow{C_SD}\|, \|\overrightarrow{C_SC'_S}\|) = 0$, see step (3) of Figure 6.5. If less than three components fulfil this condition, vector $\overrightarrow{C_SC''_S}$ with the next smallest magnitude is selected as the new seed and the process is repeated, this time with C'_S removed from G. If, however, three or more components are found, these form the desired copying subgroup with the head of the list being the new template and the translation calculated as multiples of it, see step (5) of Figure 6.5. This subgroup is removed from G and the algorithm is recursively repeated until there is not enough components left or all possible seed distances have been exhausted. This algorithm, therefore, identifies repeated copying sequences that are likely to occur by copying the same component in a row. Figure 6.8 shows an example.

6.3.3 Timeline compression

In the tested Medieval dataset shown in Figure 6.7, there were 510 separate components across the sequence, yet these formed only 17 self-similar component groups altogether. In addition, many editing operations were repeated across multiple components such as is the case of instancing, while others were independent of each other. Thus, the goal of the timeline compression is to simplify the apparent complexity of the matrix Φ while preserving the essence of the reverse engineered provenance. This can be achieved via two analytically independent collapsing steps as depicted in Figure 6.6.

Row-wise collapse. Instanced duplicates are by definition created using the same operations as their templates. Hence, a *row-wise collapse* merges all instances into their parental components while remaining components are left unmodified. This significantly reduces the matrix height, in the case of the Medieval dataset from 189 to 28 rows.

Column-wise collapse. Operations in the neighbouring frames that do not affect the same correspondence flow can be considered independent. Therefore, it is possible to perform a *column-wise collapse* given the operations at t_i do not collide with those at t_{i-1} and vice versa, i.e. when operations do not occur in neighbouring frames concurrently.

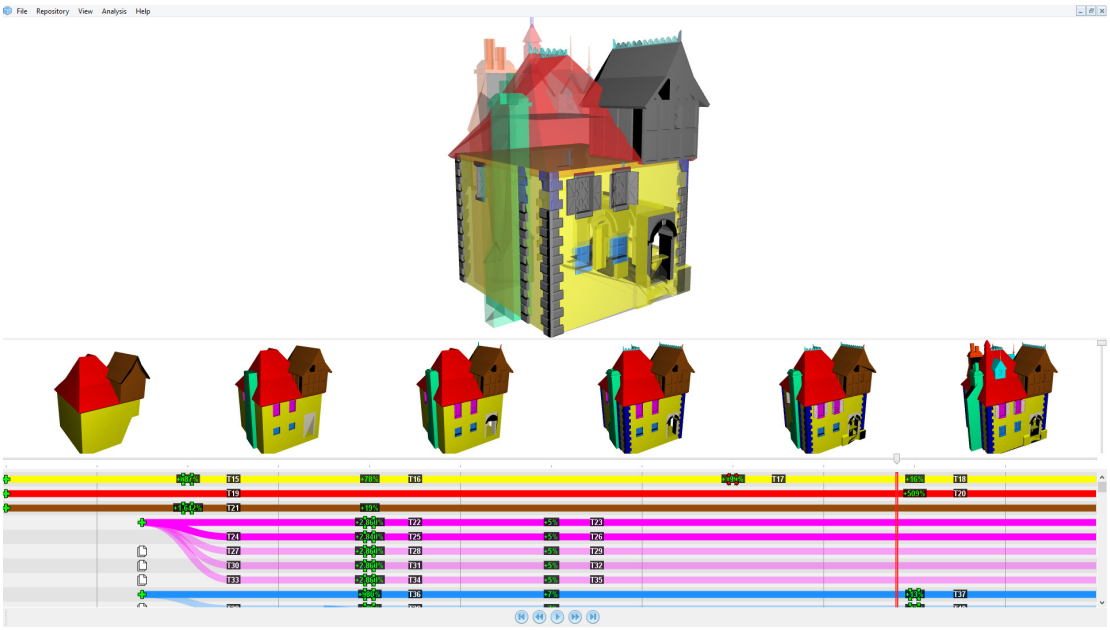


Figure 6.7: *Prototype 3D Timeline GUI implemented in a cross-platform framework Qt. Morph window is at the top, keyframes with correspondence in the middle and estimated timeline at the bottom.*

6.4 Prototype Implementation

Representing events in a timeline is a common way of abstracting complex temporal interactions into a meaningful and easily understandable flow. Apart from linear dependencies, timelines can also display hierarchical information [SNF10] and can be even used as a collaboration platform [BBB+10]. In computer graphics, however, timelines are mostly used for animation compositing such as is the case of many 3D authoring tools, e.g. Autodesk Maya [Pal13], Blender [Bla12], etc. In this implementation, a hierarchical timeline representation has been chosen as it matches the linear succession of the input data but still enables display of dependency relationships between components and their groups. Such a visualisation encourages both manual exploration and automated playback.

Similarly to desktop prototypes from Chapters 3 and 4, the viewer, shown in Figure 6.7, was implemented in a cross-platform UI framework Qt [BS08]. The ASSIMP library [SGK+14] was used to load various 3D file formats and GLC Lib [Rib14] provided the rendering capability. Although the highly parallelised pre-processing algorithm from Section 6.2 might be suitable for a GPU implementation, in this case a multi-threaded CPU version was developed as the GPU is already occupied by a large number of 3D scenes that are being rendered simultaneously. In a way similar to [DKP11], the 3D Timeline GUI shows a main blending preview at the top and a sequence of thumbnail 3D models ordered from left to right underneath. In addition, the reverse engineered provenance is visualised at the very bottom. Models are initially coloured based on their independent segmentation using a random colouring scheme, but once the correspondence estimation has been completed, the colours become consistent with the extracted timeline. In this application, it is possible to select a single component or groups of components in order to highlight the corresponding parts across the frames. The 3D thumbnails can be navigated synchronously while the main top 3D window can be explored independently of other models.

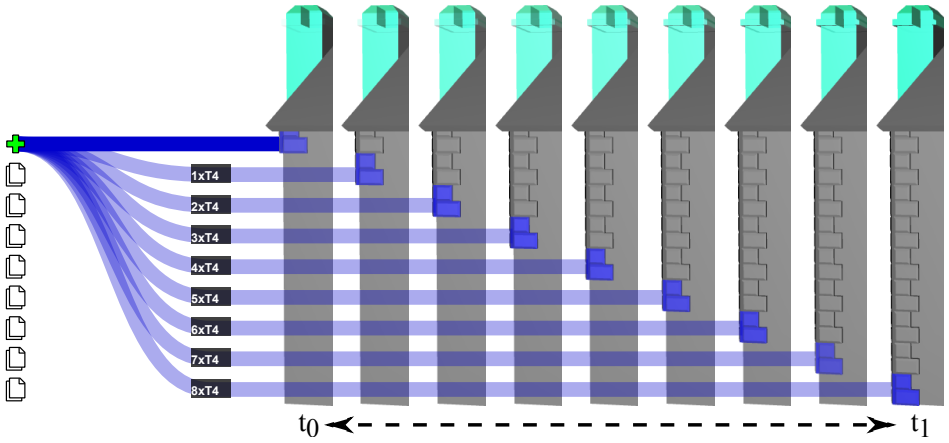


Figure 6.8: *Repeated copying blending is interpolated sequentially. In this case, it is detected on the corner stones of the Medieval dataset, see Figure 6.7.*

6.4.1 Timeline interface

The timeline itself is divided into equally spaced buckets that illustrate the time that elapsed between neighbouring keyframes, see vertical lines under each 3D thumbnail in Figure 6.7. This is only an approximation as there is no requirement for the input models to be created in equal amounts of time. In addition, each of the alternating rows signifies a correspondence flow as extracted in Section 6.3. The life-span of components is visualised as a collection of cubic Bézier curves [PT97] forming a path that can diverge from the assigned timeline row whenever a duplication has been detected. These are laid out from top to bottom in the order of a template followed by repeated copies and general duplicates. Their colouring is consistent with the correspondence assignment in the thumbnail 3D views. Instanced duplicates have decreased opacity for easier identification even when the timeline is not collapsed, as shown in Figure 6.7. Double clicking on a path highlights the corresponding components in the thumbnail views. Vectorised icons of the detected editing operations, listed in §6.3.1, are placed directly on top of the paths. From left to right, the additions are placed first, followed by relative changes in polycount and size, with translations being the last between pairs of 3D models. Such an arrangement ensures that in a duplication, the relative geometric changes appear only once in the timeline while the translations are recorded along their matching paths. Even though there is no evidence that the events have happened in any particular order, this layout significantly declutters the interface.

Just like in standard 2D animations, the time between two successive keyframes is linearly interpolated so that components at t_j are morphed into their known states at t_{j+1} via extracted editing operations. In addition, the opacity of a component $C_{i,j-1}$ changes from 100% to 0% while for $C_{i,j}$ it changes at the same rate but in reverse. Those components that do not change are rendered grey to make the immediate modifications stand out during playback. Repeated copies, however, change their opacity one by one for a pleasing visualisation of their successive construction, see Figure 6.8 for an example. Even though the animation is linearly interpolated, more emphasis can be put on certain operations by slowing down the playback speed for desired event classes. The same effect can be achieved by manually dragging the timeline slider forwards or backwards.

Dataset	Frames	Polycount	Comp.	Corr. [*ms*]	Analysis [*ms*]	Total [*s*]	Through. [*C*/*s*]
Medieval	6	16,005	510	51	40	0.09	5,604
Brick	16	16,703	141	47	25	0.07	1,958
Engine	55	3,414,103	2,460	1,435	512	1.95	1,264
Cruciform	74	924,123	23,695	74,712	1,140	75.85	312
Portico	158	2,442,104	3,622	1,908	784	2.70	1,346
Character	9	7,609,539	189	6,685	1,875	8.56	22

Table 6.1: *Statistics for test sequences evaluated on a Thinkpad X230 with Intel Core i7-3520M CPU at 2.90 GHz with 16 GB RAM on Windows 8. From left to right: the number of frames in a sequence, the cumulative number of polygons, the number of detected components, duration of the correspondence estimation, duration of the timeline analysis, sum of correspondence and analysis, and the throughput as the number of components processed per second (Components / (Correspondence + Analysis)* $\times 1000$*).*

6.5 Evaluation

The prototype 3D Timeline tool developed in Section 6.4 was evaluated on a variety of modelling sequences created by multiple professional artists in Autodesk Maya, Pixologic ZBrush, Trimble Sketchup, Blender and Luxology Modo authoring tools. Each generated sequence provided a distinct set of challenges including detailing, large number of polygons and even organic sculpting, see Appendix E. On average, they represented changes recorded every five minutes, although this was not a strict prerequisite. Table 6.1 lists the statistics for these sequences while Figure 6.9 shows the extracted timelines as well as some of the input models with their assigned correspondences. In this evaluation, $\alpha = 0.3$ was used in Equation 6.4 and a zero tolerance threshold $\sigma = 0.0001$ was set in the repeated copying detection in §6.3.2. Note that even if the actual editing history existed, it would represent only one plausible evolution that achieves the same sequences, hence no ground truth comparison was attempted. Instead, the focus was on the feasibility of the solution and its ability to aid its users in identify the key editing operations.

All measurements were performed on a Thinkpad X230 with Intel Core i7-3520M CPU at 2.90 GHz with 16 GB RAM running Microsoft Windows 8. The tested modelling sequences varied significantly in their number of input keyframes as well as scene and geometric complexity to make sure they properly represented a cross section of some of the most common editing operations. As shown in Table 6.1, the performance of the prototype system largely depends on the number of polygons as well as the number of components and their structural organisation within the scene. Although the Cruciform dataset in comparison to Portico was only half as complex, it took 28 times longer to process. This is because the Cruciform had by far the most separate components and, therefore, the correspondence estimation had to compare significantly higher numbers of possible candidates. The situation was further complicated by the presence of many repeated components such as window frame dividers that accounted for the majority of scene complexity throughout the sequence. On average, all other sequences were examined in under three seconds. The Character dataset, shown in Figure 6.1, further reveals that large polycount can also have an adverse effect on the processing speed, see the throughput column in Table 6.1. Such a behaviour is caused by the PCA bounding box calculation in §6.2.2 which takes into account all available vertices. Even though it would be possible to subsample the meshes, this technique was not employed in the tested prototype. Nevertheless, the segmentation was already parallelised on a per mesh basis, while the PCA hierarchy was calculated across all keyframes simultaneously.

		Multiview					Timeline				
	Experience	Q1	Q2	Q3	$t[m]$	SUS	Q1	Q2	Q3	$t[m]$	SUS
P1	Intermediate	F	F	F	5.9	70.0	F	T	T	3.3	67.5
P2	Expert	T	T	T	1.8	75.0	T	T	T	2.5	37.5
P3	Intermediate	T	T	F	3.1	75.0	F	T	F	2.0	72.5
P4	Intermediate	F	F	T	2.3	67.5	F	T	F	4.3	67.5
P5	Expert	F	F	F	3.6	44.5	T	T	F	1.5	77.5
P6	Intermediate	F	F	T	4.3	47.5	T	F	F	2.6	60.0
P7	Intermediate	F	F	F	2.4	37.5	T	T	F	1.9	85.0
P8	Intermediate	F	F	T	1.5	37.5	F	T	F	1.5	87.5
	Average	25%	25%	50%	3.1	56.8	50%	88%	25%	2.4	69.4

Table 6.2: *Pilot user study results based on 3 quiz questions. Evaluated are the multiview and timeline interfaces, time to completion in minutes and the system usability scale (SUS) scores. T = true, F = !T.*

Figures 6.1 and 6.9 depict the compressed results from the 3D modelling sequences. Each input set produced a consistent correspondence and a timeline visualisation, although the Portico models caused the system to lose track in four instances that are visible as steps at the bottom of Figure 6.9. This is because the façade was modelled as a single continuous mesh with drastic changes in geometry. Furthermore, in larger datasets, it became obvious from the timeline that many components, once created, do not change throughout the rest of their life-span. This indicates selective modelling where the detail is progressively added rather than performing adjustments on a global scale. It is, this, believed that 3D Timeline provides useful insights into large sequences that was not achievable otherwise, see §6.5.1.

6.5.1 User study

The usability of the 3D Timeline prototype was evaluated in a comparative user study with eight post-graduate researchers in the field of computer graphics. These were selected based on their intermediate or expert level in 3D modelling experience. In order to compare and contrast the timeline interface, a *multi-view* scenario with only a basic side by side visualisation and interlinked 3D navigation was used in the study. The participants were given a sample dataset before each trial to familiarise themselves with the different visual environments. The task was to answer a short quiz with regards to the modelling provenance. The questions asked where the most unique components were added, how many components had the longest lifespan and which models had the most duplication. After each session, a system usability scale (SUS) [Bro96] questionnaire was handed out, see Appendix D. The order of the datasets and interfaces was shuffled according to Latin square to prevent any effects of learning, see §4.5.1.

As shown in Table 6.2, the overall average success of answering the quiz questions correctly based on predetermined ground truth was 54% in the timeline versus 33% in the multiview scenario. The timeline also scored better in terms of the system usability scale (SUS) reaching a grade B, i.e. an above the average user interface for the task at hand. According to this score, the users found it easy to interact with the interface, would not need any technical assistance and would happily use the 3D Timeline tool again. One expert user, however, assigned a low SUS to the timeline due to his strong preference of the multiview interface. Note that the study had only six frames to give the baseline viewer a fair comparison. The general experience is that on longer sequences of 15+ frames, visual examination without the timeline becomes nearly impossible due to a vast number of changes being present.

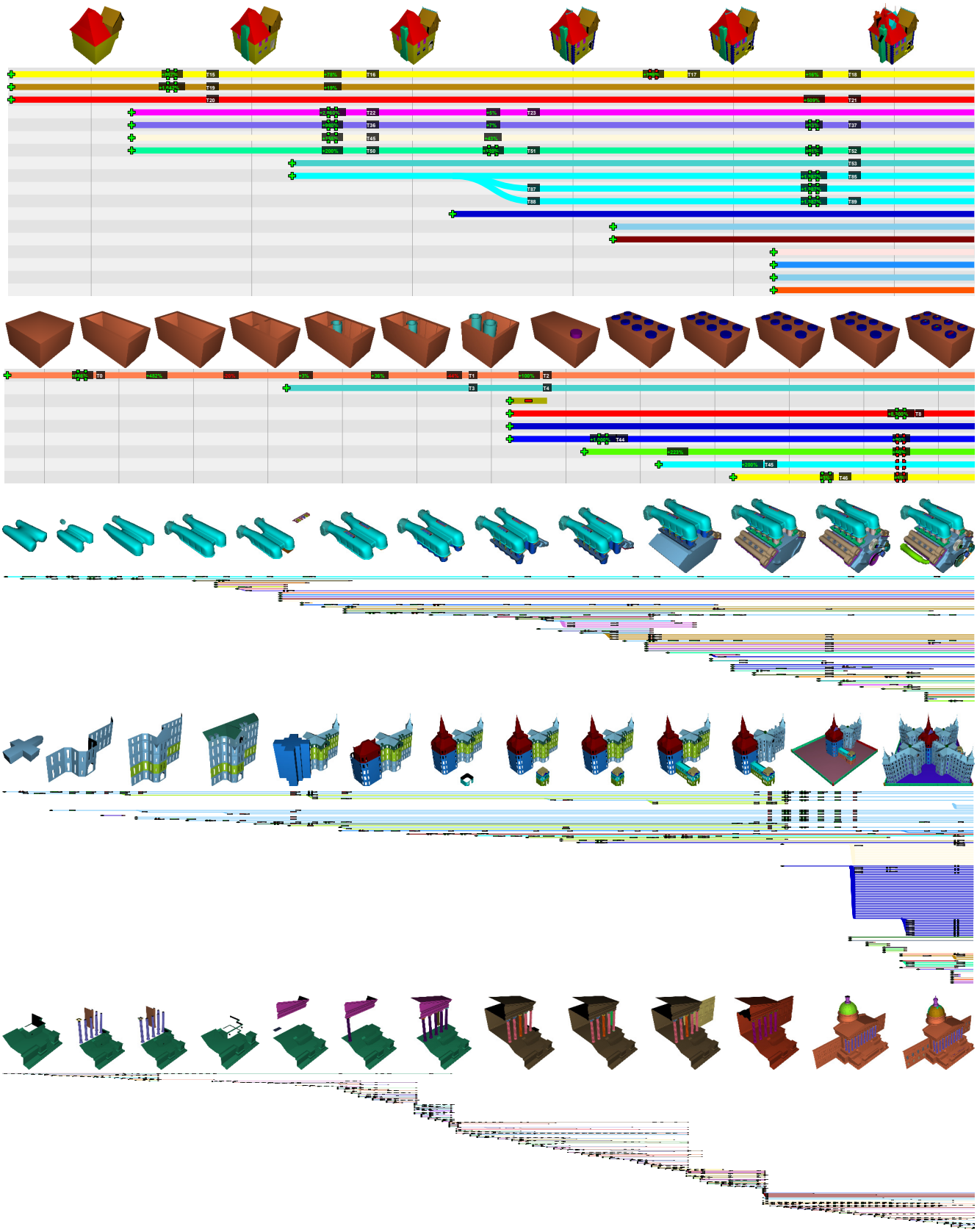


Figure 6.9: *3D Timeline results. Collapsed timelines for Medieval, Brick, Engine, Cruciform and Portico datasets as presented in Table 6.1. Character sequence is shown in Figure 6.1. Note that for brevity only subsets of keyframes are shown for the Engine, Cruciform, and Portico datasets.*

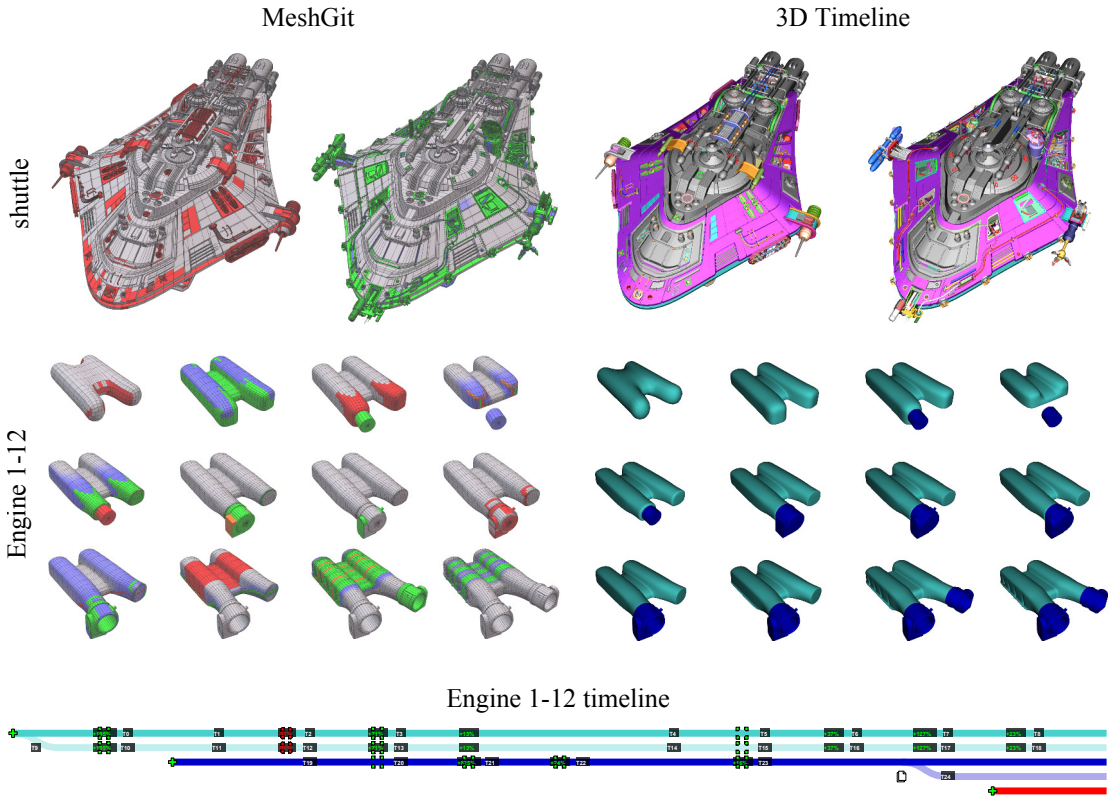


Figure 6.10: *3D Timeline vs. MeshGit [DP13] comparison. Compared is their* `shuttle` *model and the first 12 frames of our* `Engine` *dataset. MeshGit shows changes in adjacency (red/green), geometry (blue) and sequential changes (orange). The timeline shows unmodified component groups (grey) and modified (corresponding colours). It also detects bending between Engines 8 and 9.*

6.5.2 MeshGit comparison

The MeshGit system by Denning and Pellacini [DP13] focuses on vertex-level pairwise model comparisons, hence a complementary reverse engineering subproblem. This system can be run in a sequence to provide a form of linear differencing as shown in Figure 6.10. Such an approach is similar to a 2-way sequential differencing already described in Section 4.2. Being higher-level and component-based, however, the detection of 3D Timeline is faster and scales to much larger datasets. For example, on the *shuttle*, their largest example with over 5,000 separate components, MeshGit takes 9.7 minutes to complete. When the same model pair was evaluated on a comparable hardware, described at the beginning of Section 6.5, it took mere 14 seconds in 3D Timeline.

On datasets from Figure 6.9 where there were strong changes in adjacency of vertices and faces, MeshGit matched large areas that do not correspond as repositioned, added or deleted. See the first 12 models of the Engine dataset in Figure 6.10. 3D Timeline can also handle modifications that translate components significantly, the main limitation discussed in the MeshGit paper. Nevertheless, in many cases MeshGit provides insights into fine-grained editing that 3D Timeline does not attempt to recover due to tractability in real-life applications. Despite this, it is conceivable to use 3D Timeline for component-based analysis and then selectively apply MeshGit to vertex-level changes once the correspondence has been established. For instance, this could be activated on user selection.

6.6 Discussion

Since 3D Timeline makes no assumptions about the temporal coherence of the modelling effort, nor does it rely on any scene graph organisational structures or revision metadata, it presents a practical solution to a complex reverse engineering problem. It thus gives its users the ability to explore some of potentially many high-level semantic operations in large 3D datasets that would not be explained by a VCS. This demonstrates that it is possible to imply editing operations even when no metadata is available as stated by the third research Question in Chapter 1.

In general, 3D Timeline highlights deformations and other modifications that are hard to detect by visual inspection alone. This becomes even more apparent with the growing size of the input sequences, although even for pairs of polygonal 3D models this solution already reveals previously unnoticed editing operations. For instance, between the first two keyframes of the Medieval dataset, there is a vertical stretch of the façade geometry with an upward translation in its roof structures. Even in a direct side by side comparison, it is difficult to notice this, yet both the timeline and the blending preview reveal this precisely. See the detected translation on a red line between the first two frames in Figure 6.7. At close inspection of the Brick and Engine datasets in Figure 6.9, it can be also seen that the respective logos were created separately and put in place once completed. If the frames are visually too close to each other, i.e. nearly perfectly identical in their geometry, the algorithm detects no changes and those frames are collapsed. If, however, they are too dissimilar, at some point the correspondence estimation fails and the components are marked as deleted in one frame and added in the subsequent frame.

Another interesting revelation arises from the Portico dataset that consists of 158 distinct models, the largest sequence tested with the system. A common practice when creating massive 3D models is to temporarily disable or even totally remove certain parts of a scene in order to lower its memory footprint, hence unburden the editor. This hidden geometry is then reintroduced at a later stage when the full scene is required. As visible in the first 29 keyframes, represented by seven green thumbnails at the beginning of the Portico dataset, the last sequence in Figure 6.9, the modeller created a set of pillars with attempted detailing which then disappear entirely. Later, the columns reappear, but this time with a different height and shape. Unlike the common practice of hiding geometry, the 3D Timeline identified this situation as a massive deletion followed by an addition. This is because the reintroduced geometry significantly differs from that of the supposed original. Upon confirmation with the modeller it became obvious that it was indeed the intention to delete the first version of the columns so that the positioning and the height of the roof would govern their construction in the next stages. Without the timeline it would have been very difficult if not impossible to discover such cases. The prototype application was even able to render the largest sequences, although the frame rate would drop when navigating all 158 Portico models together.

6.6.1 Limitations

While the centre of this work is the analysis and visualisation only, once a timeline of this type is available, it would be possible to start editing the sequence based on its history. For example, parts of a 3D scene could be substituted with their previous revisions for quick alterations. Similarly, the timeline interface could be used to extract editing provenance directly from a version controlled repository. However,

systems such as 3D Repo use DAGs to represent revision history. Unlike standard trees, DAGs cannot be simplified into a collection of linear construction sequences because one model can have multiple parents in a DAG. Nevertheless, 3D Timeline can be used on liner histories stored in 3D Repo. Furthermore, the system in its current form cannot imply the relative ordering of operations between two frames. For example, it cannot reliably establish whether an increase in the number of polygons has happened before or after a duplication. This would require an extensive study of modelling behaviour based on software instrumentation which might provide evidence for patterns of events that occur more frequently than others. Similarly, the choice of a template component in duplication is currently arbitrary, although, it might be possible to devise extra set of rules to break ties. In addition, 3D Timeline does not detect join operations, i.e. multiple components becoming a single manifold surface, or mirroring, rotations and scaling. This is noticeable in the largest dataset where the most of the geometry is modelled as a single continuous mesh. Although the correspondence estimation is greedy, hence not globally optimal, it is robust in most cases and together with the detection of changes can uncover even non-rigid transformations such as bending, see the Engine model in Figure 6.10. Nevertheless, the estimation might fall into a local minima. In general, these cases are resolved by a majority vote reassignment within a self-similar group as depicted in Figure 6.3. In rare cases, however, if the PCA-aligned bounding boxes of non-corresponding components are too similar, the algorithm might be unable to recover and would result in a wrong assignment. Furthermore, the tight threshold of the repeated copying detection does not support approximate regularities, unlike [PMW+08].

6.7 Chapter Summary

The third and last research question in Chapter 1 asked whether a plausible provenance can be extracted from a sequence of 3D models. Hence, 3D Timeline presented in this chapter is a novel tool for reverse engineering of a part-based provenance from linearly consecutive polygonal models. In comparison to previous works described in Chapter 2, this approach does not require any pre-recorded editing operations from the actual modelling session. Thus, it is applicable to all sorts of legacy datasets that are often found across various industries. For instance, 3D Repo, introduced in Chapter 3, can preserve changes in 3D scenes that are discovered using the 3D Diff tool from Chapter 4. In order to establish part-based correspondence in two models, 3D Diff relies on revision metadata such as the unique component names or IDs values to be present in the input data formats. These are decomposed so that their delta changes can be stored in a NoSQL database. Such an approach, however, supports only tracking of simple additions, deletions and modifications. In contrast, the 3D Timeline is able to reliably determine many more high-level semantic operations that are likely to have happened during the editing. To achieve this, the system establishes correspondence based on contextual support from hierarchies of scene components that are independent of their original scene graph structures. Hence, the timeline takes as input only a sequence of polygonal 3D models consisting of polygonal meshes. The pre-processing step in Section 6.2 establishes the self-similar groups in each of the models and then propagates a consistent correspondence across the entire construction chain. In Section 6.3, such correspondence flows are gathered into a sparse matrix which is then semantically analysed for modifications across components. Apart from the stan-

dard additions and deletions, these include changes in the size and polycount, duplication, instancing and repeated copying. Finally, the results of such an analysis are visualised as a timeline so that the matching components share the same colouring throughout the sequence. In order to reduce the apparent complexity of the timeline, two independent compression operations have been introduced in §6.3.3. These significantly lower the number of displayed rows but still preserve the essence of the reverse engineered provenance. If the input 3D models are too similar, i.e. no changes have been detected, those can be hidden, too. Therefore, such a novel system addresses the aforementioned research question.

To evaluate the feasibility of the proposed solution, the timeline was implemented into a prototype tool in Section 6.4 that offers a main 3D blending preview at the top, thumbnail 3D models in the middle and an extracted timeline at the bottom. In Section 6.5, the tool was successfully tested on six construction sequences spanning architectural modelling, CAD prototyping and even free form sculpting. The pilot user study in §6.5.1 suggests that this tool is usable by untrained users who preferred it over a standard side by side visualisation technique that is commonly found in existing modelling packages. Thus, the 3D Timeline can be useful for artists revisiting modelling processes to learn from. The conclusion is that it is indeed possible to extract high-level semantic editing operations even when revision metadata is not available what is especially useful for legacy datasets. The hope is that this type of UI will inspire software vendors who could embed such a system into their editors in the future.

Chapter 7

Conclusions

In the domain of computer graphics, collaborative editing and visualisation are becoming increasingly important. Studied were topics such as generation of tutorials from photo editing sessions based on author demonstrations [GAL+09] and even integrated revision control of 2D images [CWC11]. These approaches, however, require instrumentation of the authoring tools and tracking of the individual editing operations. Similarly, the visualisation and exploration of document histories [GMF10] requires video recording of the entire sessions and a large disk space to store the generated sequences. Nevertheless, this particular system was extended to support recording in CAD software under the official name Autodesk Screencast [Aut12]. Furthermore, in the context of 3D models, recording of editing operations was used to enable clustering and visualisation of modelling histories [DKP11]. Even commercial products such as the VisTrails [Vis12] plug-in for Autodesk Maya preview only recorded edits. However, most of the time such a capture might not exist. Only very recently the extraction of semantic meaning from pairs of modified 2D images [HXM+13] and vertex-based comparison of 3D models [DP13] were proposed.

Thus, the research in this thesis explored the management and visualisation of non-linear history of polygonal 3D models. As argued in Chapter 1, there is the need for a highly scalable and extensible system that is able to support the widest possible range of polymorphic assets in a unified version control and online visualisation system. Hence, the newly proposed 3D Repo system from Chapter 3 was designed in such a way that is independent of any specific modelling paradigm. Chapter 4 further documented the highly related concept of visual 3D differencing and merging in order to support asynchronous collaborative 3D editing directly through this framework. The aim of the accompanying user study was to establish a suitable visualisation strategy for the system as several different possibilities were identified. Chapter 5 further explored the suitability of the Representational State Transfer (REST) [FGM+99] style of architecture for the purposes of 3D version control directly over the Internet. Its experimental evaluation tested six different encoding formats for their efficiency and speed of delivery. Finally, Chapter 6 introduced a novel algorithm for reverse engineering of a part-based provenance from 3D models. This was visualised in an interactive timeline to enhance the higher-level semantic understanding of such sequences. The findings from a user study suggest that this type of UI is preferred over a standard side by side visualisation. Thus, the overarching goal of the thesis was to design and demonstrate all components of a domain-specific version control framework applicable to the purposes

of collaborative 3D editing of large scale engineering 3D models. The importance of this work was stressed on numerous examples throughout the text. This closing chapter, therefore, summarises the research presented in this thesis. Firstly, a conclusion is presented relating the chapters back to the research questions from Chapter 1. Next, the results from each chapter are recounted. Finally, directions for potential future work are outlined especially in the areas that were outside the scope of this thesis.

7.1 Contributions

The overarching goal of the research was to investigate and propose a highly extensible framework for management of non-linear history of various types of 3D assets that would enable collaborative 3D modelling on the scale required by the industry. Chapter 1, therefore, introduced three central research questions that define the components of this goal as follows:

1. *Can asynchronous collaborative 3D editing be scaled up to useful model sizes via application of a domain specific version control system?*
2. *Can such a specific 3D versioning framework deliver real-time visualisations of large scale 3D scenes over the Internet?*
3. *Can a plausible editing provenance be extracted from a sequence of 3D models?*

The first two questions are concerned with the management of and distributed access to 3D revisions in a remote cloud-based repository. Both deal with the central premise whether a domain specific version control framework can address the shortcomings of a file-based 3D asset management and whether such a collaborative framework would be suitable for online repository visualisation. The final question is concerned with legacy datasets that were not recorded using the newly developed framework. This thesis thus made theoretical as well as practical contributions. The theoretical contributions consist of a proposal for a novel architecture for version control of 3D assets and the associated processes and algorithms. The practical contributions concern demonstration of the feasibility of the proposed solutions via several different prototype implementations and their corresponding evaluations.

7.1.1 Theoretical contributions

This section presents substantive theoretical contributions documented throughout Chapters 3–6. These concern the design of a novel architecture capable of representing, storing and distributing different types of 3D assets independent of any specific authoring tools or file formats. The feasibility of this approach was established on several prototype implementations that, as a part of the practical contributions, are further summarised in §7.1.2. Hence, the theoretical contributions address the main research questions.

The first question asked whether a domain specific VCS can support asynchronous collaborative 3D editing of useful model sizes, i.e. those sizes that are commonly found in the industry. A central observation of this research is that large 3D scenes consist of thousands of disparate components organised in hierarchical data structures. This is certainly the case for most authoring tools, 2D and 3D file formats as well as rendering engines [FJJ00, BO04, SML06]. Such an observation is further supported by numerous examples of large engineering 3D models presented in Chapters 3 and 6. The next

prominent observation is the striking similarity between a scene graph and a revision history both of which can be modelled after a DAG. This enabled the definition of a novel 3D repository in a NoSQL DB. Unlike their relational predecessors, the document-oriented NoSQL DBs provide a dictionary-like interface where each polymorphic document is stored as a collection of key-value pairs independent of other documents. Hence, the repository can represent not only various types of 3D assets but also their associated non-linear history in a single centralised location. To futureproof the design, a nondescript scene graph node was selected as the smallest unit of change. Thus, each part of a 3D scene is treated as an opaque binary document, the granularity of which is decided by the individual authoring tool that created it. Since the version control logic only depends on the structural organisation of the data blocks and not their content, the repository can handle any asset type. This is because it is the responsibility of the client application to interpret such documents while the framework is being oblivious. These concepts apply to all linear data stores and not just NoSQL DBs. A subquestion of the first research question in Chapter 1 further asked whether it is possible to sustain asynchronous collaborative 3D editing without the need for per asset locking. Thus, the problem of conflict resolution suitable for 3D assets was addressed by Chapter 4. The newly introduced concept of visual 3D differencing and merging was compared to an analogous situation in software engineering. Apart from the standard 2-way and 3-way differencing, Section 4.2 further proposed the notions of explicit and implicit conflicts that are specific to 3D models. Based on these results, the first research question may be answered affirmatively as the proposed framework successfully decoupled 3D modelling from its long-term storage and provided a suitable means of scalable asynchronous collaborative 3D editing not possible otherwise.

The second question asked whether it is possible to distribute real-time visualisations of 3D scenes stored in such a domain-specific VCS. To achieve this, two different approaches were identified as subquestions of this research question. Firstly, it might be possible to connect to the repository directly to query individual revisions. Secondly, an indirect connection via a gateway service might enable retrieval of data representations that are not explicitly stored in the repository. The first approach requires the client applications to decode a prescribed data format into individual 3D assets that can then be reconstructed into a scene graph representation before rendering. Three such clients were demonstrated in Chapter 3. However, memory limited devices, e.g. mobiles and tablets, might not be able to render whole revisions from the repository. Examples of these are many large engineering 3D models that are the main target of this research. Thus, an important feature of the system is its ability to track and serve, apart from whole revisions, also their subsets. This is possible because of the introduction of a modified materialised paths notation in §3.2.3 in Chapter 3 which enables scene changes to be recorded without the need to reindex any previously stored entries. Nevertheless, to make this design independent of the underlying data store, the version control logic has to reside at the application level. Hence, Chapter 5 further introduced a server-side daemon service capable of enforcing the internal rules of the repository on all incoming requests. The newly proposed REST architecture enables client applications to query 3D resources in a representation that is most suitable to their specific needs. This preference can be based on the network properties or the processing capabilities of the receiving client. Therefore, the

second research question can be answered affirmatively since the design of the newly proposed VCS for 3D assets enables direct as well as indirect online distribution of 3D assets. Its feasibility is further supported by real-time demonstrators summarised as practical contributions in §7.1.2. These include desktop, web and mobile clients from Chapter 3 as well as a client-server architecture from Chapter 5.

The third question asked whether it is possible to imply the editing history from a sequence of 3D models especially when no revision metadata is available. This question is important as there are a large number of legacy datasets that were not recorded using the newly developed system from the previous two research questions. Such datasets arise because many popular modelling packages auto-increment and auto-save the work in progress, but do not provide any further support for their understanding or management. Chapter 6 investigated novel means of reverse engineering of the editing provenance from consecutive 3D models. This is related to another contribution of this thesis which proposed sequential visual 3D differencing in Chapter 4. However, 3D differencing identifies scene components only as added, deleted, modified, unmodified and conflicted based on the requirements of the VCS introduced in Chapter 3. Reverse engineering aims to provide further high-level explanation of the semantic operations that created the resulting set of models. Thus, Section 6.1 proposed a tractable multi-stage algorithm that can identify a plausible editing history. In a pre-processing stage, the algorithm first segments the 3D scenes into separate components and then estimates their correspondence within each model individually, i.e. by finding self-similar groups of components, as well as across the entire sequence. Next, the identified correspondence flows are semantically analysed for changes in their size and polycount, translations, duplication, instancing and repeated copying. Finally, the resulting provenance is visualised as an interactive timeline. This is further supported by experimental implementation and evaluation as discussed in §7.1.2. The only requirement of this implementation is a known temporal ordering of the input 3D models. Nevertheless, it would be easy to imply such an ordering if this was not available as, in general, the models tend to grow in complexity over time. Since no further assumptions about the models including their internal structure or temporal coherence of the editing effort have been made, the third research question can be answered affirmatively.

7.1.2 Practical contributions

In addition to the main theoretical contributions, this thesis also aimed to investigate the practical implications of the management and visualisation of non-linear history of 3D models. Prototype implementations developed throughout Chapters 3–6 demonstrate the feasibility of the proposed solutions that address the individual research questions as defined in Chapter 1.

Chapter 3 introduced a novel NoSQL DB schema that was developed to evaluate the data organisation approach proposed in Section 3.2. This schema prescribes several compulsory fields that are shared by every document regardless of the asset type it represents. These include the unique identifier (UID) and shared identifier (SID) values both represented as universally unique identifier (UUID) [Tel08], materialised paths, the type of the document as well as its encoding API level. Additional optional fields define all of the individual properties of cameras, comments, materials, meshes, revisions, textures and transformations. Large data blocks such as vertices, normals and face indices as well as bitmap images

are stored as serialised binary entries. Such a definition is capable of tracking millions of polygons spread across thousands of components as demonstrated in Section 3.5. To support this, three different client applications, developed in Section 3.4, connect to the remote 3D repository directly. A desktop-based client was implemented in C++, a cross-platform UI framework Qt [BS08], the Open Asset Import Library (ASSIMP) [SGK+14] and the OpenGL Library Class (GLC Lib) [Rib14]. This enables loading of more than 40 popular 3D file formats into a unified in-memory scene graph representation that is transformed into the required DB schema for storage. This multi-threaded implementation supports full version control, rendering of hundreds of simultaneous 3D contexts with interlinked navigation and parallelised repository management. Furthermore, a web browser-based client was developed as a combination of a Java applet, JavaScript and WebGL [Mar11] in order to visualise the contents of the repository over the Internet. This, same as all the other clients, interprets the DB schema via a dedicated core library. Finally, an Android app was developed in Java and OpenGL ES [ML10] to demonstrate the suitability of the system for the purposes of public consultation. Users of the app can view 3D revisions and submit localised comments for further analysis.

Chapter 4 presented a prototype 3D Diff tool capable of 2-way and 3-way visual 3D differencing and merging. The tool supports multiple visualisation modes as well as the detection of explicit and implicit conflicts as defined in Section 4.2. This was implemented as part of the desktop client from Chapter 3, hence, it is based on C++, Qt, ASSIMP and GLC Lib. It can, therefore, load various types of 3D models to find their discrepancies and propose automatic merge suggestions. The addition of the common ancestor of the differenced models further helps to resolve otherwise ambiguous cases when it would not be possible to establish whether a component has been added, deleted or modified concurrently. Correspondence between the components was obtained from revision metadata stored in a form of SID values introduced in Chapter 3. Their equality was based on an early reject byte-by-byte comparison. Various visualisation modes were evaluated in a formative user study in Section 4.5, findings of which suggest that the most preferred UI is a hybrid visualisation where the differenced 3D models are visible alongside the proposed merge result. In addition, the detection of implicit conflicts was found useful, although the implementation only supported detection of bounding box intersections which, in some cases, was not detailed enough. Thus, future tools should concentrate on mesh intersections.

Chapter 5 defined a novel API for version control of 3D assets over the Internet. This API supports retrieval of collections of resources, single resources and even their specific attributes in an encoding representation that is the most suitable for the receiving client. A prototype sever-side daemon application was developed in node.js [Ihr13] and a JavaScript port of the core version control library from Chapter 3. The corresponding web browser-based client was developed in XML3D [SKR+10], JavaScript and WebGL. Similarly to the base repository implementation, the UID and SID values were realised using UUIDs that can be uniquely generated without any centralised coordination. Several different encodings were tested in order to evaluate the system. These include XML, JSON, BSON [Mon14a], SIG [BJFS12], OpenCTM [Gee09] and ArrayBuffers [BJFS12]. While the XML and JSON formats can utilise native parsing capabilities of modern web browsers, the remaining encodings required develop-

ment of custom decoders. Section 5.5 evaluated all of them for their efficiency and speed of delivery. Although the text-based formats provide the least gain in terms of size, they can be compressed using standard compression methods in order to achieve data sizes similar to the binary formats. However, the native parsing as well as the lower number of required XHR requests make them faster throughout the experiments when comparing to other evaluated formats. Nevertheless, the choice of the best encoding largely depends on the network properties and the decoding capabilities of the receiving clients.

Chapter 6 presented a prototype 3D Timeline tool for reverse engineering of a part-based provenance from consecutive 3D models. This tool, same as the desktop client from Chapter 3 and the 3D Diff tool from Chapter 4, was developed in C++, Qt, ASSIMP and GLC Lib. Such a setup enables comparison of otherwise incompatible file formats. In a pre-procesing stage, the tool automatically segments the loaded models, establishes their component-based correspondence and evaluates it in order to visualise the implied editing timeline. This is rendered using Bézier curves below the main blending preview and thumbnail 3D models. Each component is colour-coded to match other corresponding components across the thumbnails as well as their detected life-spans and operations in the timeline. The apparent complexity of the timeline is further reduced by analytically independent row-wise and column-wise collapse. Similarly to standard 2D animations, the timeline playback is linearly interpolated between the keyframes. This means that the opacity of the immediately modified components changes as the time progresses. Furthermore, the timeline can be explored manually by scrubbing the cursor forwards and backwards in time. The accompanying user study demonstrated that such a GUI is preferred over a standard multi-view interface by scoring higher in the system usability scale (SUS) [Bro96] questionnaire.

7.2 Results

Each of the Chapters 3–6 presented a number of results that are summarised below. These include novel technical approaches and prototype implementations in support of the proposed framework. The theoretical and practical contributions are summarised in Section 7.1.

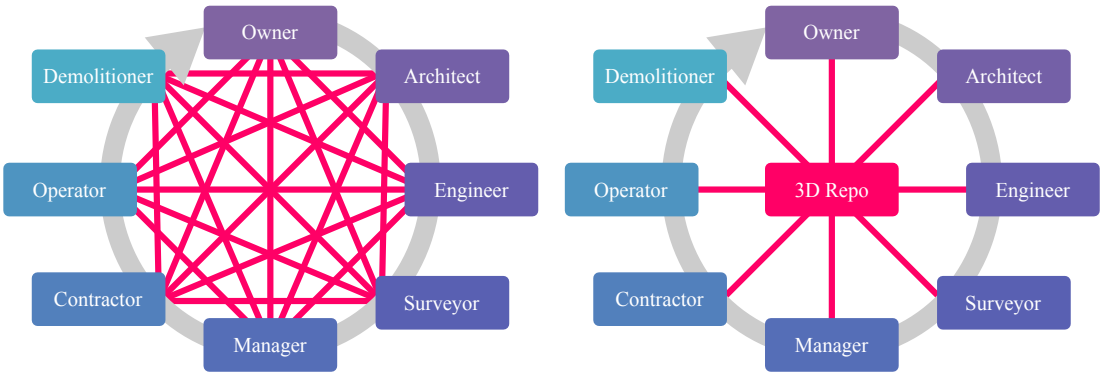
3D version control database. The architecture of a version control framework specifically designed for 3D assets was reported in Chapter 3. This explored the asynchronous collaboration and storage of decomposed polygonal 3D scenes in a remote NoSQL database. In order to support the vast number of authoring tools and packages that already exist, the exchange of information was facilitated via external 3D files. Since a DB acts as a remote repository, such asset files are now considered only temporary data representations that are transformed into a unified DB schema for eventual permanent storage. Individual assets are encoded as Binary JSON (BSON) documents before being uploaded to the repository. Upon retrieval, such documents are decoded and reconstructed into a meaningful scene graph representation for immediate rendering or further editing. Each document is considered a delta increment in this system. Although a single vertex modification would force an entire mesh document to be committed again, this is still considerably smaller than the whole 3D scene, which might be composed of tens of thousands of components. Such a scene configuration is anyway decided by the authoring tool that created it. This chapter further explored the similarities between a scene graph and a revision history and introduced

a novel extension to the materialised paths notation in order to efficiently represent DAGs in a linear data store. Three different DB front-end applications were developed to support the non-linear version control, automated merging as well as visualisations over the Internet. These prototypes are able to represent polygonal geometry together with other types of assets alongside their revision history in a single centralised location. The framework was demonstrated on millions of polygons and hundreds of different revisions that each can be accessed remotely.

Visual 3D differencing and merging. The notion of visual 3D differencing and merging was introduced in Chapter 4. This is required in order to support asynchronous collaborative 3D editing without the need for per asset locking. When multiple users edit the same part of a 3D scene simultaneously, the newly developed 3D Diff tool identifies changes between the revisions and offers automated merge suggestions. For this, new concepts of conflict classification have been introduced. On the one hand, explicit conflicts are detected whenever discrepancies amongst the 3D models arise. On the other hand, implicit conflicts based on the bounding box intersections signify cases where the semantics of the models have been violated. These are often caused as a side effect of the merging process itself. By further integrating the knowledge about the common ancestor of the differenced models, it is possible to resolve otherwise ambiguous cases. Several common visualisation strategies have been examined. The results of a preliminary user study suggest that the most preferred visualisation is the one with the main merge result visible alongside the two smaller differenced 3D models.

XML3DRepo daemon service. A novel client-server Representational State Transfer (REST) architecture able to deliver and render revisions directly from a remote 3D repository was presented in Chapter 5. This system serves representations of 3D assets that are independent of the underlying storage or file formats. Data from the repository can be queried as collections of resources, single resources and even separate attributes depending on the specific requirements of the receiving client. Such an architecture enabled implementation of six significantly different encoding formats. These were evaluated for their speed and efficiency across three popular web browsers. The accompanying experimental evaluation demonstrated that the API provides a consistent way of addressing version controlled 3D assets over the Internet. It further established that none of the tested encoding formats for 3D data on the web would be suitable for all scenarios and applications. While some formats offered considerable size reduction via compression, they were slow to decompress on the client-side. Other formats, although faster, would often require more HTTP requests that would impair the overall performance of the system. Hence, the best format must be selected depending on the network properties and the client's processing capabilities.

3D timeline reverse engineering. Finally, an approach for reverse engineering of editing provenance was reported in Chapter 6. This implies high-level semantic operations from legacy datasets that were not recorded in a VCS such as the one presented in Chapter 3. The algorithm first automatically segments the input 3D models and established a component-based correspondence across the entire collection. This is subsequently analysed for editing operations including additions, deletions, changes in their size and polycount, duplication, instancing and even repeated copying. Such operations are then visualised in an interactive timeline which encourages automated as well as manual exploration of the construction



(a) *Before* **(b)** *After*

Figure 7.1: *Construction supply chain data exchange complexity.* **(a)** *In the construction industry, there is a high level of complexity when exchanging any kind of data, especially 3D, amongst the members of the supply chain.* **(b)** *In the future, 3D Repo will aim to reduce the need for handovers and formal Request for Informations (RFIs) via application of a domain-specific version control repository. Similar parallels can be found in the aerospace, automotive and creative industries where 3D models are heavily relied upon, too.*

sequence. However, it provides only one plausible history as many different actions might have resulted in the same 3D sequence. A comparative user study further demonstrated that such a GUI is preferred to a standard multi-view visualisation that is commonly found in existing modelling packages. Hence, it can be useful for trying to identify occasions when important changes took place in a large data collection.

In principle, even higher-order representations such as parametrised models, non-uniform rational basis spline (NURBS), constructive solid geometry (CSG) or boundary representation (BREP) that are commonly found in CAD modelling could be version controlled in the newly developed framework as long as they are assigned to specific nodes within the overall scene hierarchy. The main advantage is that the VCS can store not only control points and surface boundaries but also their pre-calculated attributes such as centre of gravity, volume and mass. However, by design, the system is not able to support any specific queries about spatial occupancy, clash detection, etc., thus unable to perform integrity checks.

7.3 Directions for Future Work

There are many avenues for future work. Some of these were already discussed in the context of limitations and potential benefits of the proposed approach, while others cover those topics that are outside the scope of this thesis. The intention is to continue developing this new architecture into a scalable platform that can be used by the community, thus making all of the developed software and specifications open source and available on GitHub. The framework will hopefully facilitate crowd-sourcing of 3D models for various purposes in the near future. In the upcoming releases, the system will be expanded to support standard animations, bones and various types of engineering metadata such as volumes, materials, etc. It will also support all attributes of the Industry Foundation Classes (IFC) [bui13] data format, the de facto standard of the construction industry. The aim is to define a process change for the whole construction supply chain as depicted in Figure 7.1. Due to the use of decentralised UUIDs, it will be also possible to extend the framework into a fully distributed system where the clients connect to a local

database which is then dynamically synchronised with a remote repository. The overall aim is to track five dimensional (5D) data, i.e. *3D models* with *cost* over *time*.

The main architecture overview in Chapter 3 suggested that it would be possible to avoid the use of intermediary GUI applications in order to connect to the remote repository directly. In a production environment, it might be necessary to develop a plug-in to one or more popular editing packages. Such a plug-in would be able to exploit the available information about the immediate changes. For example, the system would know exactly when an asset has been created or modified and these changes could then be seamlessly committed back to the repository for safekeeping without the need for any kind of user intervention. Hence, the internal workings of the version control would be entirely hidden from the end user. This approach could also support access control via locking. Although the support for locking can be easily added to the DB schema, it would be difficult to enforce when dealing with 3D models at the level of individual files. Thus, plug-ins might be better suited for this task, too, since they can represent much richer information during runtime. Nevertheless, it would be still necessary to establish the granularity of the locks. For instance, the entire components or assemblies could be locked, or a polyline could specify an interface beyond which no changes are permitted. Such locks could have time-limited validity which, upon expiration, would be automatically released.

Another feature that is already planned for the upcoming release is the ability to search for 3D geometry based on spatial queries. Meshes stored in the system already represent bounding box coordinates including their centroids. Given MongoDB's built-in geospatial indexing, it would be possible to facilitate out-of-core rendering directly over the Internet. This technique is popular with GIS platfroms that often span vast land areas that would not fit into the operational memory [LP02, PG07]. There, the data is organised on local disks in such a way that it is possible to fetch only the closest proximity geometry required for immediate rendering. A similar approach would be possible with the version control repository proposed in this thesis. This would, however, require a development of a novel 3D protocol for a real-time online visualisation as discussed in §2.4.3 in Chapter 2. Data formats such as X3D and its DOM integration model X3DOM are already evaluating compression streaming over the network. Recent developments in the field include POP Buffers [LJBA13] and the Shape Resource Container (SRC) data node [LTBF14]. Ultimately, the aim is to scale the framework up to millions of concurrent users contributing to the same 3D scene which will eventually be composed of billions or even quadrillions of polygons. This will require addressing practical limits of the 3D data storage in databases such as clustered server-side infrastructure, tiling of the world when viewing and editing, etc.

Furthermore, the visual 3D differencing and merging of 3D models in Chapter 4 imposed certain restrictions about the granularity of the changes that can be detected. To match the overall framework requirements outlined in Chapter 3, the 3D Diff tool supports change detection at the level of individual scene graph nodes. Since the tool only establishes whether there are any discrepancies between corresponding nodes, it can be easily extended to take into account changes beyond the currently supported node types. For example, existing image-based differencing techniques could be added in order to identify which portions of a texture were modified. Similarly, once the modified meshes have been found,

it would be possible to further compare individual vertices. The next big challenge will be searching random collections of unrelated 3D models to find the correspondences and then to express exactly what is different. First steps towards this goal were already achieved in Chapter 6.

The results in Chapter 5 demonstrate that amongst the most popular tested web data formats there is currently no single 3D encoding that would fit all devices, networks and applications. This is because of the existing limitations in terms of compression, decoding speed and the number of HTTP requests that greatly influence the performance of such a client-server infrastructure. Hence, there is the need for a dynamic system that would be able to automatically establish the best encoding format based on some predetermined heuristics. Alternatively, it might be possible to develop a new format that would be generic enough to represent all kinds of 3D assets, yet efficient in its encoding. This would also require the ability to support progressive loading. Currently, the system from Chapter 5 renders individual components as soon as they become available giving the user an immediate visual feedback. In the future, this approach could be combined with vertex-level morphing and quantisation in order to visualise massive data instantly after the very first few bytes have been loaded. Nevertheless, large 3D models tend to have too many components that the existing JavaScript engines simply cannot handle. Hence, another suggestion for future work is the investigation of techniques that can merge meshes together for visualisation, but still support per component selection. In addition, it will be interesting to evaluate the best approach of data upload into such a system. For example, the data might be decomposed directly in the web browser before committing. However, given the assumption that there is always going to be many more reads than writes, it might be equally feasible to upload raw 3D models for a more powerful server to process. Similarly to the version control repository in Chapter 3, it would be further possible to extend the proposed REST API in Section 5.2 with the functionality to support spatial and proximity-based queries. If the repository contained the semantic meanings of individual scene components, these, too, could be queried. It is likely that, in the future, it will be possible to enable collaborative 3D editing directly in web browsers, just like Google Docs did for text documents [Vie09].

Finally, a simple addition to the 3D Timeline tool from Chapter 6 would be the application of a heat map to visualise the rate of change on a morphed 3D model itself. In a way similar to the 3D Diff from Chapter 4, once the component-based correspondence has been established, it might be possible to combine the existing tool with further techniques that are able to determine vertex-level differences. This could achieve a form of deep semantic understanding that would be used for training purposes when the editing session could not have been recorded. An interesting avenue for future work is also the exploration of automated intention preservation while modifying the timeline. It might be possible to replace individual components along the timeline while preserving the semantic meaning of other related changes within the sequence. Another addition would be finding structural regularities in 3D models, e.g. [PMW+08, MPWC13], that could automatically suggest improvements and fixes in otherwise incomplete or damaged sequences of input files that often span from laser scanning.

It is sincerely hoped that these results will inspire software vendors to embed such solutions in their future modelling packages.

Appendices

Appendix A

Publications

The following publications, all appearing in peer-reviewed international conferences, are presented in chronological order according to date of publication. Where appropriate, the section in this thesis corresponding to the work presented in the publication is referenced.

Capturing Time-of-Flight Data with Confidence

Malcolm Reynolds, Jozef Doboš, Leto Peel, Tim Weyrich and Gabriel J. Brostow

IEEE Computer Vision and Pattern Recognition (CVPR) 2011

```
@inproceedings{Reynolds:2011:CTD:2191740.2191922,
  author = {Reynolds, M. and Dobo\v{s}, J. and Peel, L. and Weyrich, T. and Brostow, G. J.},
  title = {Capturing Time-of-Flight Data with Confidence},
  booktitle = {Computer Vision and Pattern Recognition},
  series = {CVPR '11},
  year = {2011},
  isbn = {978-1-4577-0394-2},
  pages = {945--952},
  numpages = {8},
  url = {http://dx.doi.org/10.1109/CVPR.2011.5995550},
  doi = {10.1109/CVPR.2011.5995550},
  acmid = {2191922},
  publisher = {IEEE Computer Society},
  address = {Washington, DC, USA}
}
```

Revision Control Framework for 3D Assets

Jozef Doboš and Anthony Steed

Eurographics 2012 Posters

Features extracts of work presented in Chapter 3.

```
@inproceedings{posters:1-2:2012,
  crossref = {posters-proc},
  author = {Jozef Dobo\v{s} and Anthony Steed},
  title = {Revision Control Framework for 3D Assets},
  pages = {1-2},
  URL = {http://diglib.eg.org/EG/DL/conf/EG2012/posters/001-002.pdf},
  DOI = {10.2312/conf/EG2012/posters/001-002}
}
@proceedings{posters-proc,
  editor = {Andrea Fusiello and Michael Wimmer},
  title = {EG 2012 - Posters},
```

```
  year = {2012},
  isbn = { -},
  issn = {1017-4656},
  address = {Cagliari, Sardinia, Italy},
  publisher = {Eurographics Association}
}
```

3D Revision Control Framework

Jozef Doboš and Anthony Steed

17th International Conference on 3D Web Technology (Web3D) 2012

Features extracts of work presented in Chapter 3.

```
@inproceedings{Dobos:2012:RCF:2338714.2338736,
  author = {Dobo\v{s}, Jozef and Steed, Anthony},
  title = {3D Revision Control Framework},
  booktitle = {Proceedings of the 17th International Conference on 3D Web Technology},
  series = {Web3D '12},
  year = {2012},
  isbn = {978-1-4503-1432-9},
  location = {Los Angeles, California},
  pages = {121--129},
  numpages = {9},
  url = {http://doi.acm.org/10.1145/2338714.2338736},
  doi = {10.1145/2338714.2338736},
  acmid = {2338736},
  publisher = {ACM},
  address = {New York, NY, USA}
}
```

3D Diff: An Interactive Approach to Mesh Differencing and Conflict Resolution

Jozef Doboš and Anthony Steed

ACM SIGGRAPH 2012 Talks

Features extracts of work presented in Chapter 4.

```
@inproceedings{Dobos:2012:DIA:2343045.2343064,
  author = {Dobo\v{s}, Jozef and Steed, Anthony},
  title = {3D Diff: An Interactive Approach to Mesh Differencing and Conflict Resolution},
  booktitle = {ACM SIGGRAPH 2012 Talks},
  series = {SIGGRAPH '12},
  year = {2012},
  isbn = {978-1-4503-1683-5},
  location = {Los Angeles, California},
  pages = {15:1--15:1},
  articleno = {15},
  numpages = {1},
  url = {http://doi.acm.org/10.1145/2343045.2343064},
  doi = {10.1145/2343045.2343064},
  acmid = {2343064},
  publisher = {ACM},
  address = {New York, NY, USA}
}
```

Visualizing 3D Models in Aid of Public Consultation

Jozef Doboš, Alvise Simondetti and Anthony Steed

ACM SIGGRAPH Asia 2012 Symposium on Apps

Features extracts of work presented in Chapter 3.

```
@inproceedings{Dobos:2012:VMA:2407696.2407705,
  author = {Dobo\v{s}, Jozef and Simondetti, Alvise and Steed, Anthony},
  title = {Visualizing 3D Models in Aid of Public Consultation},
  booktitle = {SIGGRAPH Asia 2012 Symposium on Apps},
  series = {SA '12},
  year = {2012},
  isbn = {978-1-4503-1916-4},
  location = {Singapore, Singapore},
  pages = {9:1--9:1},
  articleno = {9},
  numpages = {1},
  url = {http://doi.acm.org/10.1145/2407696.2407705},
  doi = {10.1145/2407696.2407705},
  acmid = {2407705},
  publisher = {ACM},
  address = {New York, NY, USA}
}
```

3D Diff: An Interactive Approach to Mesh Differencing and Conflict Resolution

Jozef Doboš and Anthony Steed

ACM SIGGRAPH Asia 2012 Technical Briefs

Features extracts of work presented in Chapter 4.

```
@inproceedings{Dobos:2012:DIA:2407746.2407766,
  author = {Dobo\v{s}, Jozef and Steed, Anthony},
  title = {3D Diff: An Interactive Approach to Mesh Differencing and Conflict Resolution},
  booktitle = {SIGGRAPH Asia 2012 Technical Briefs},
  series = {SA '12},
  year = {2012},
  isbn = {978-1-4503-1915-7},
  location = {Singapore, Singapore},
  pages = {20:1--20:4},
  articleno = {20},
  numpages = {4},
  url = {http://doi.acm.org/10.1145/2407746.2407766},
  doi = {10.1145/2407746.2407766},
  acmid = {2407766},
  publisher = {ACM},
  address = {New York, NY, USA},
  keywords = {3D Diff, conflict resolution, indirect conflict, merging}
}
```

XML3DRepo: A REST API for Version Controlled 3D Assets on the Web

Jozef Doboš, Kristian Sons, Dmitri Rubinstein, Philipp Slusallek and Anthony Steed

18th International Conference on 3D Web Technology (Web3D) 2013

Features extracts of work presented in Chapter 5.

```
@inproceedings{Dobos:2013:XRA:2466533.2466537,
  author = {Dobo\v{s}, Jozef and Sons, Kristian and Rubinstein, Dmitri
            and Slusallek, Philipp and Steed, Anthony},
  title = {XML3DRepo: a REST API for Version Controlled 3D Assets on the Web},
  booktitle = {Proceedings of the 18th International Conference on 3D Web Technology},
```

```
  series = {Web3D '13},
  year = {2013},
  isbn = {978-1-4503-2133-4},
  location = {San Sebastian, Spain},
  pages = {47--55},
  numpages = {9},
  url = {http://doi.acm.org/10.1145/2466533.2466537},
  doi = {10.1145/2466533.2466537},
  acmid = {2466537},
  publisher = {ACM},
  address = {New York, NY, USA},
  keywords = {3D repo, CRUD, REST, XML3D, revision control}
}
```

3D Timeline: Reverse Engineering of a Part-based Provenance from Consecutive 3D Models

Jozef Doboš, Niloy J. Mitra and Anthony Steed

Proceedings of Eurographics 2014, Computer Graphics Forum, Volume 33, No. 2

Features extracts of work presented in Chapter 6.

```
@article{Dobos:2013:XRA:2466533.2466537,
  author = {Dobo\v{s}, Jozef and Mitra, Niloy J. and Steed, Anthony},
  title = {3D Timeline: Reverse Engineering of a Part-based Provenance
           from Consecutive 3D Models},
  journal = {Comp. Graph. Forum},
  year = {2014},
  location = {Strasbourg, France},
  numpages = {10},
  volume = {33},
  number = {2},
  publisher = {John Wiley & Sons Ltd},
  address = {New York, NY, USA}
}
```

Appendix B

List of Acronyms

The following acronyms appear in this thesis:

2D	two dimensional
3D	three dimensional
5D	five dimensional
3DR	3D Repository by the Advanced Distributed Learning
3D Repo	3D Repository
ACID	atomicity, consistency, isolation and durability
AEC	architecture, engineering and construction
AJAX	Asynchronous JavaScript and XML
AMD	Advanced Micro Devices, Inc.
API	application programming interface
ASSIMP	Open Asset Import Library
BEM	Built Environment Modelling
BIM	building information modelling
BITE	Breakthrough Information Technology Exchange
BREP	boundary representation
BSON	Binary JSON
BSP	binary space partitioning
CAD	computer-aided design
CAE	computer-aided engineering

CAM	computer-aided manufacturing
CATIA	Computer-aided Three-dimensional Interactive Application
CDE	Common Data Environment
CDN	content delivery network
CentOS	Community Enterprise Operating System
COLLADA	Collaborative Design Activity
CSG	constructive solid geometry
CPU	central processing unit
CSCW	computer-supported cooperative work
CSS	cascading style sheets
CVE	collaborative virtual environment
CVS	Concurrent Versions System
CRUD	create, read, update and delete
DAG	directed acyclic graph
DB	database
DBMS	database management system
DCCP	Datagram Congestion Control Protocol
DFKI	German Research Centre for Artificial Intelligence
DFS	distributed file system
DIVE	Distributed Interactive Virtual Environment
DOM	document object model
EPSRC	Engineering and Physical Sciences Research Council
X3D	Extensible 3D
FI	FastInfoset
FS	file system
GB	Gigabyte
GHz	Gigahertz

GIS	Geographic Information System
GIMP	GNU Image Manipulation Program
GLC Lib	OpenGL Library Class
GoD	Gaming on Demand
GPU	graphics processing unit
GUI	graphical user interface
GUID	globally unique identifier
HAR	HTTP Archive
HCI	human-computer interaction
HDRI	high-dynamic-range imaging
HS2	High Speed 2
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HUT	Helsinki University of Technology
HVAC	heating, ventilating, and air conditioning
ID	identifier
IDE	integrated development environment
IFC	Industry Foundation Classes
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
KB	kilobyte
KTH	The Royal Institute of Technology
LIFO	last in, first out
MASSIVE	Model, Architecture and System for Spatial Interaction in Virtual Environments
MB	megabyte
MEP	mechanical, electrical, and plumbing services

MIME	Multipurpose Internet Mail Extensions
MIT	Massachusetts Institute of Technology
MMOG	massively multiplayer online game
MR	Minimal Reality
ms	millisecond
NoSQL	No Structured Query Language
NPM	node packaged modules
NURBS	non-uniform rational basis spline
OpenCTM	Open Compressed Triangle Mesh
OpenGL	Open Graphics Library
OpenGL ES	OpenGL for Embedded Systems
OS	operating system
OSM	OpenStreetMap
OST	Olympic Stadium Transformation
PDES	Product Data Exchange Standard
PDF	Portable Document Format
PNG	Portable Network Graphics
PCA	principal component analysis
PHIGS	Programmer's Hierarchical Interactive Graphics System
PS4	PlayStation 4
RAM	random access memory
RDBMS	relational database management system
RDP	Remote Desktop Protocol
REST	Representational State Transfer
RFI	Request for Information
SID	shared identifier
SIG	Sequential Image Geometry

SIMNET Simulation Network

SRC Shape Resource Container

STCP Scalable TCP

STEP STandard for the Exchange of Product Model Data

STL STereoLithography

SUS system usability scale

SVG Scalable Vector Graphics

SVN Apache Subversion

TCP Transmission Control Protocol

UCL University College London

UDP User Datagram Protocol

UI user interface

UK United Kingdom

UN United Nations

URI uniform resource identifier

URL uniform resource locator

USC University of Southern California

UTF-8 Universal Character Set Transformation Format–8-bit

UID unique identifier

UUID universally unique identifier

VBO Vertex Buffer Object

VCS version control system

VEIV Virtual Environments, Imaging and Visualisation

VRML Virtual Reality Modeling Language

VTK Visualization Toolkit

W3C World Wide Web Consortium

WAVES WAterloo Virtual Environment System

WebGL Web Graphics Library

XHR XMLHttpRequest

XHTML Extensible HyperText Markup Language

XMI XML Metadata Interchange

XML Extensible Markup Language

XML3D Extensible Markup Language 3D

XPath XML Path Language

Appendix C

3D Diff Questionnaire for Chapter 4

3D Diff User Study

Welcome to the 3D Diff User Study. In the next 50 minutes, you will be asked to test and evaluate 4 different 3D merging techniques with varying level of automation. After each section, you will be asked to evaluate your experience.
*Required

Age *
(Enter below)

Gender *
(Choose one option)
- Male
- Female

What is your level of familiarity with 3D modelling? *
(Choose one option)
- Very familiar - took multiple training courses in the past
- Familiar - took one training course in the past
- Somewhat familiar - tried some but without any formal training
- Not familiar - no practical experience

How confident are you about your ability to manipulate an unseen 3D model using a tutorial? *
(Choose one option)
- Very confident
- Confident
- With no expectations
- Apprehensive

Testing and Evaluation

Please test each of the tools now and complete the relevant part of the questionnaire.

Your tasks is to merge edits from two different 3D models in such a way that you obtain the most visually pleasing result while making sure you preserve the latest changes whenever possible. If bounding boxes of 3D models did not intersect in either of the two versions of the models, it is undesirable that you introduce new intersections in the merged result. If you think you cannot resolve conflicting edits by selecting only one of the version, you can leave them in a conflicted state and indicate that you would need to export to an external editor such as Blender, Max or Maya.

The following colour coding is used to convey information:
- ORANGE - current selection
- RED - conflicted - change happened on the same piece of geometry and system cannot automatically decide what the result is meant to be.
- GREEN - resolved - only one of the two conflicted meshes is taken as the merged result
- BLUE - modified - only one of the the versions of a mesh has been modified
- BORDO - deleted - a mesh has been deleted
- VIOLET - added - a mesh has been added
- TURQUOISE - bounding box conflicted - bounding boxes intersect where they did not intersect before

Continue »

Powered by Google Docs

Report Abuse - Terms of Service - Additional Terms

3D Diff User Study

Part 1

Rate the accuracy of each of the following statements on a scale of 1 to 5 as per below:

1- Strongly disagree
2 - Disagree
3 - Neither agree nor disagree
4 - Agree
5 - Strongly agree

Which tool did you use?

Two-way diff Overlay ▾

The 3D merging tool is easy to use.

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

I would be able to use the tool without the initial tutorial.

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

The tool seems to be reliable.

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

I feel that the tool helped me complete the given task well.

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

The tool did NOT reduce my ability to finish the given task on time.

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

The functionality of the tool was sufficient for me to complete the given task according to the instructions.

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

I have NOT experienced any technical difficulties during the testing of the tool.

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

I trust the suggestions for automated merging the tool offers.

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

Comments

(Add any comments to Part 1 below)

« Back Continue »

3D Diff User Study

*Required

Part 2

Rate the accuracy of each of the following statements on a scale of 1 to 5 as per below:

1- Strongly disagree
2 - Disagree
3 - Neither agree nor disagree
4 - Agree
5 - Strongly agree

Which tool did you use? *

Two-way diff Standard ▾

The 3D merging tool is easy to use. *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

I would be able to use the tool without the initial tutorial. *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

The tool seems to be reliable. *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

I feel that the tool helped me complete the given task well. *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

The tool did NOT reduce my ability to finish the given task on time. *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

The functionality of the tool was sufficient for me to complete the given task according to the instructions. *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

I have NOT experienced any technical difficulties during the testing of the tool. *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

I trust the suggestions for automated merging the tool offers. *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

Comments *

(Type any comments to Part 2 below)

« Back Continue »

3D Diff User Study

*Required

Part 3

Rate the accuracy of each of the following statements on a scale of 1 to 5 as per below:

1- Strongly disagree
2 - Disagree
3 - Neither agree nor disagree
4 - Agree
5 - Strongly agree

Which tool did you use? *

Three-way diff Standard ▾

The 3D merging tool is easy to use. *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

I would be able to use the tool without the initial tutorial. *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

The tool seems to be reliable. *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

I feel that the tool helped me complete the given task well. *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

The tool did NOT reduce my ability to finish the given task on time. *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

The functionality of the tool was sufficient for me to complete the given task according to the instructions. *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

I have NOT experienced any technical difficulties during the testing of the tool. *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

I trust the suggestions for automated merging the tool offers. *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

Comments *

(Type any comments to Part 3 below)

« Back Continue »

3D Diff User Study

*Required

Part 4

Rate the accuracy of each of the following statements on a scale of 1 to 5 as per below:

1- Strongly disagree
2 - Disagree
3 - Neither agree nor disagree
4 - Agree
5 - Strongly agree

Which tool did you use? *

Three-way diff Smart ▾

The 3D merging tool is easy to use. *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

I would be able to use the tool without the initial tutorial. *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

The tool seems to be reliable. *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

I feel that the tool helped me complete the given task well. *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

The tool did NOT reduce my ability to finish the given task on time. *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

The functionality of the tool was sufficient for me to complete the given task according to the instructions. *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

I have NOT experienced any technical difficulties during the testing of the tool. *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

I trust the suggestions for automated merging the tool offers. *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

Comments *

(Type any comments to Part 4 below)

« Back Continue »

3D Diff User Study

*Required

Comparison

Please rank the four tools on their ease of use. *

	Very hard	Hard	Not sure	Easy	Very easy
Two-way diff Overlay	○	○	○	○	○
Two-way diff Standard	○	○	○	○	○
Three-way diff Standard	○	○	○	○	○
Three-way diff Overlay	○	○	○	○	○

Did you find any functionality in one of the tools particularly helpful? *
(Add comments below)

Did you find any functionality in one of the tools distracting or hard to use? *
(Add comments below)

What functionality do you think it would be useful to add? *
(Add comments below)

End

The experiment is now over. Please submit your evaluation.

Thank you very much for your help with testing the tools and for completing the questionnaire.

« Back Submit

Never submit passwords through Google Forms.

Appendix D

3D Timeline Questionnaire for Chapter 6

3D Timeline User Study

Thank you for participating in our user study. During this study you will explore two different graphical user interfaces **multiview** and **timeline** that both support exploration of 3D modelling histories. Your task is to answer very simple quiz questions with regards to presented 3D models, assess the usability of the systems and at the end compare and contrast the two interfaces and the types of explorations you will have experienced. The experiment will last no more than 15 minutes and you can withdraw at any point in which case your answers will be **invalidated**.

Q0. What is your 3D modelling experience? Please tick as appropriate.

☐ Beginner ☐ Intermediate ☐ Expert

Please continue on the next page.

Multiview User Interface

In the multiview user interface you are presented a sequence of 3D models ordered from the bottom right to the top left. You can navigate all models simultaneously and select and highlight individual meshes in each window independently.

You will now be given a sample dataset to familiarise yourself with the user interface. You can ask questions at any time.

Next you will be given a dataset on which to answer the following questions. The time it takes you to complete the task will be measured.

Dataset:

Q1. Between which two models were the most unique non-duplicate components added?

Q2. How many components have the longest life span, i.e. appear in the most models?

Q3. Between which two models the most duplication happened?

Duration:

Please continue on the next page.

System Usability Scale

If you feel you cannot respond to a particular item, you should mark the centre point of the scale.

Interface:

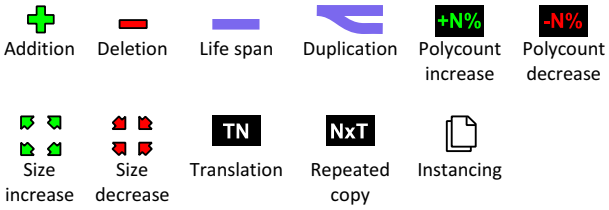
	Strongly disagree				Strongly agree
1. I think that I would like to use this system frequently.	1	2	3	4	5
2. I found the system unnecessarily complex.	1	2	3	4	5
3. I thought the system was easy to use.	1	2	3	4	5
4. I think that I would need the support of a technical person to be able to use this system.	1	2	3	4	5
5. I found the various functions in this system were well integrated.	1	2	3	4	5
6. I thought there was too much inconsistency in this system.	1	2	3	4	5
7. I would imagine that most people would learn to use this system very quickly.	1	2	3	4	5
8. I found the system very cumbersome to use.	1	2	3	4	5
9. I felt very confident using the system.	1	2	3	4	5
10. I needed to learn a lot of things before I could get going with this system.	1	2	3	4	5

Please continue on the next page.

Timeline User Interface

In the timeline user interface you are presented a sequence of 3D models ordered from left to right. You can navigate all models simultaneously and select and highlight individual meshes in each window independently. In addition, you can move the slider to reveal the flow of editing operations in the main morphing window. Tool tips on each 3D view reveal the file name of each model.

Timeline legend:



You will now be given a sample dataset to familiarise yourself with the user interface. You can ask questions at any time.

Next you will be given a dataset on which to answer the following questions. The time it takes you to complete the task will be measured.

Dataset:

Q1. Between which two models were the most unique non-duplicate components added?

Q2. How many components have the longest life span, i.e. appear in the most models?

Q3. Between which two models the most duplication happened?

Duration:

Please continue on the next page.

System Usability Scale

If you feel you cannot respond to a particular item, you should mark the centre point of the scale.

Interface:

	Strongly disagree				Strongly agree
1. I think that I would like to use this system frequently.	1	2	3	4	5
2. I found the system unnecessarily complex.	1	2	3	4	5
3. I thought the system was easy to use.	1	2	3	4	5
4. I think that I would need the support of a technical person to be able to use this system.	1	2	3	4	5
5. I found the various functions in this system were well integrated.	1	2	3	4	5
6. I thought there was too much inconsistency in this system.	1	2	3	4	5
7. I would imagine that most people would learn to use this system very quickly.	1	2	3	4	5
8. I found the system very cumbersome to use.	1	2	3	4	5
9. I felt very confident using the system.	1	2	3	4	5
10. I needed to learn a lot of things before I could get going with this system.	1	2	3	4	5

Please continue on the next page.

Open Ended Questions

Q1. How would you compare the two tools?

Q2. Which tool would you prefer and why?

Q3. Any additional comments?

Thank you for your participation, the user study is now over.

Appendix E

3D Timeline Input Models for Chapter 6

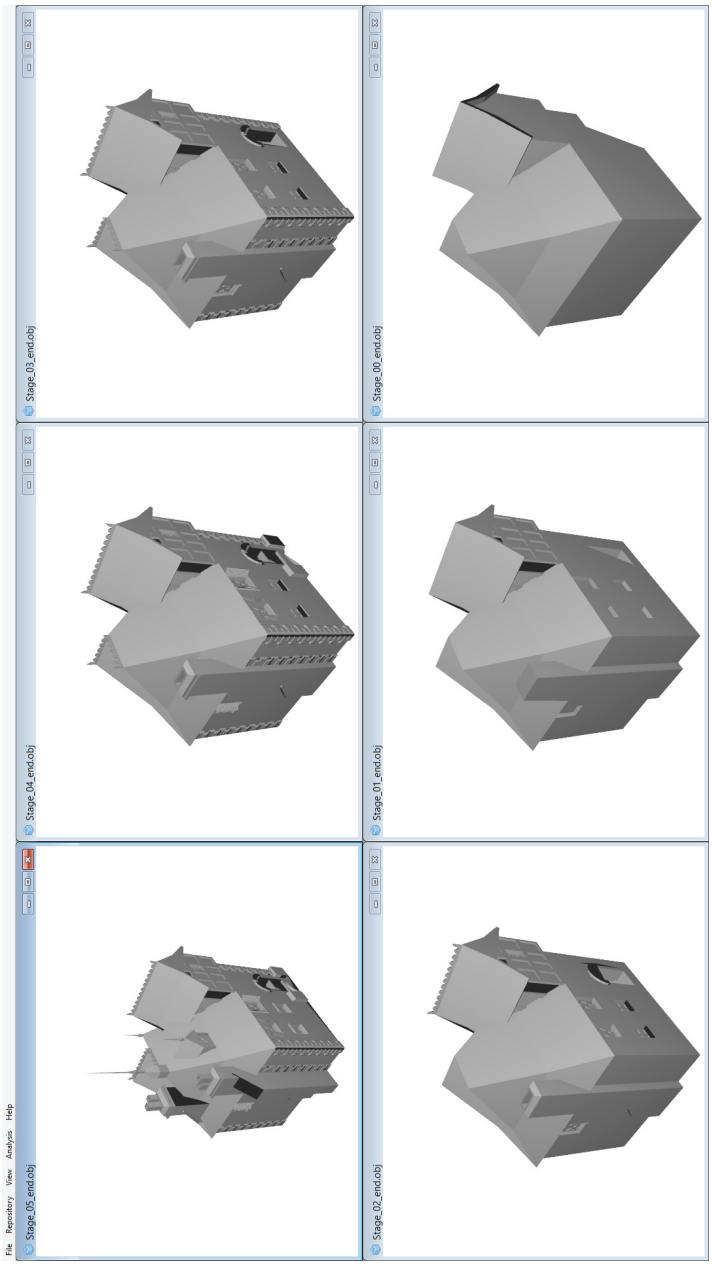


Figure E.1: *Medieval dataset.*

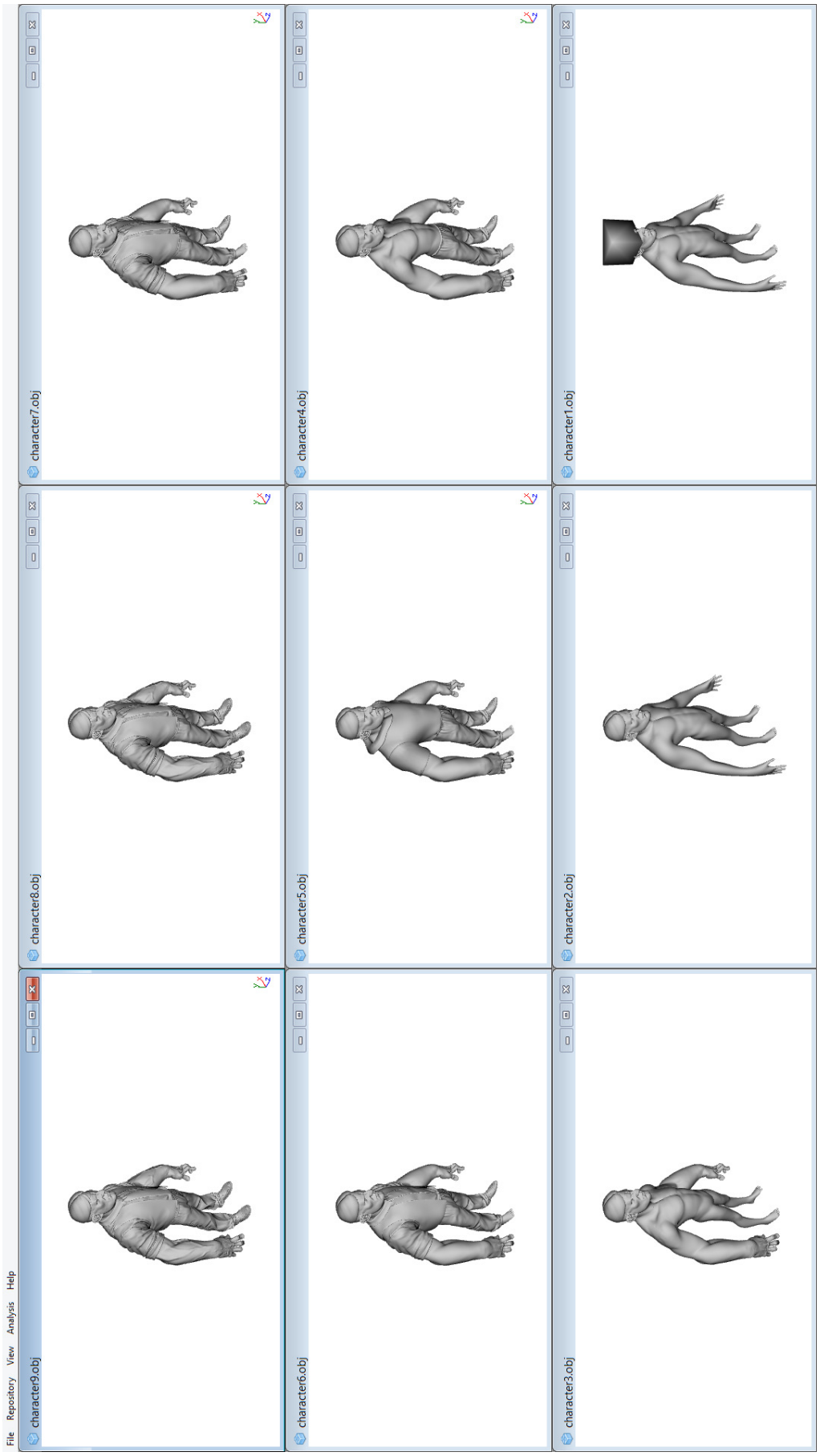


Figure E.2: *Character dataset.*

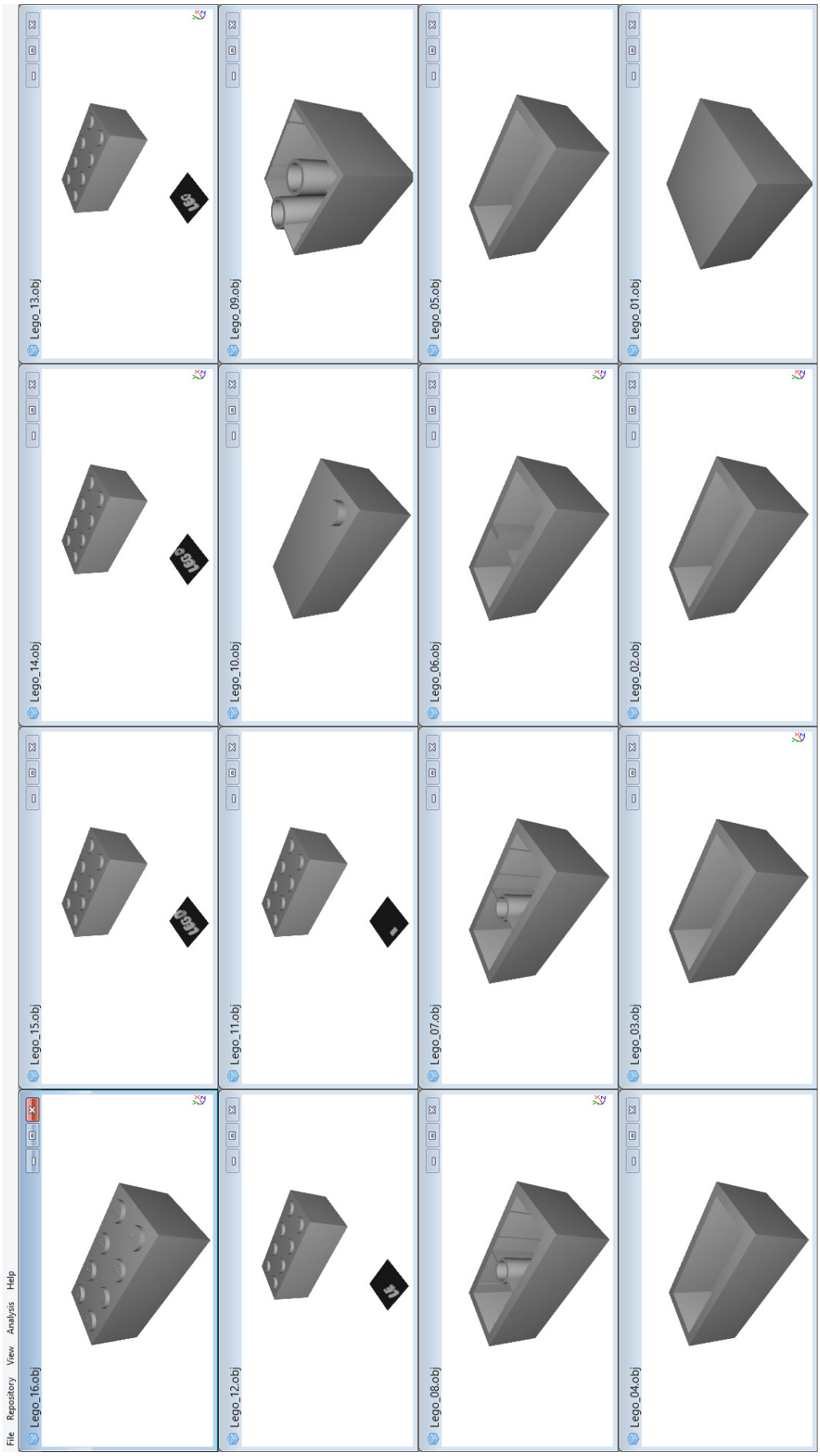


Figure E.3: *Brick dataset.*

Figure E.4: *Engine dataset.*

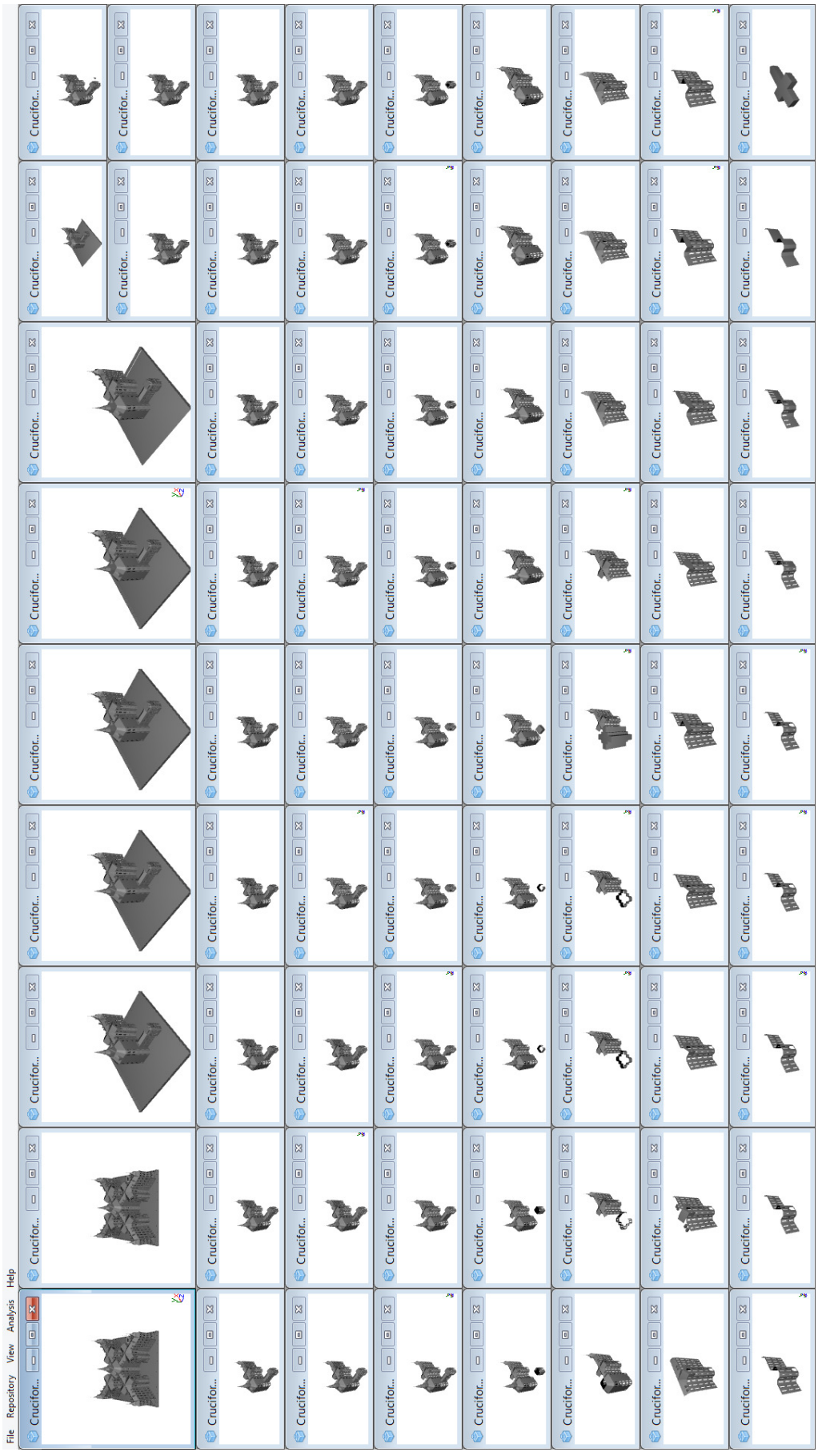


Figure E.5: *Cruciform dataset.*

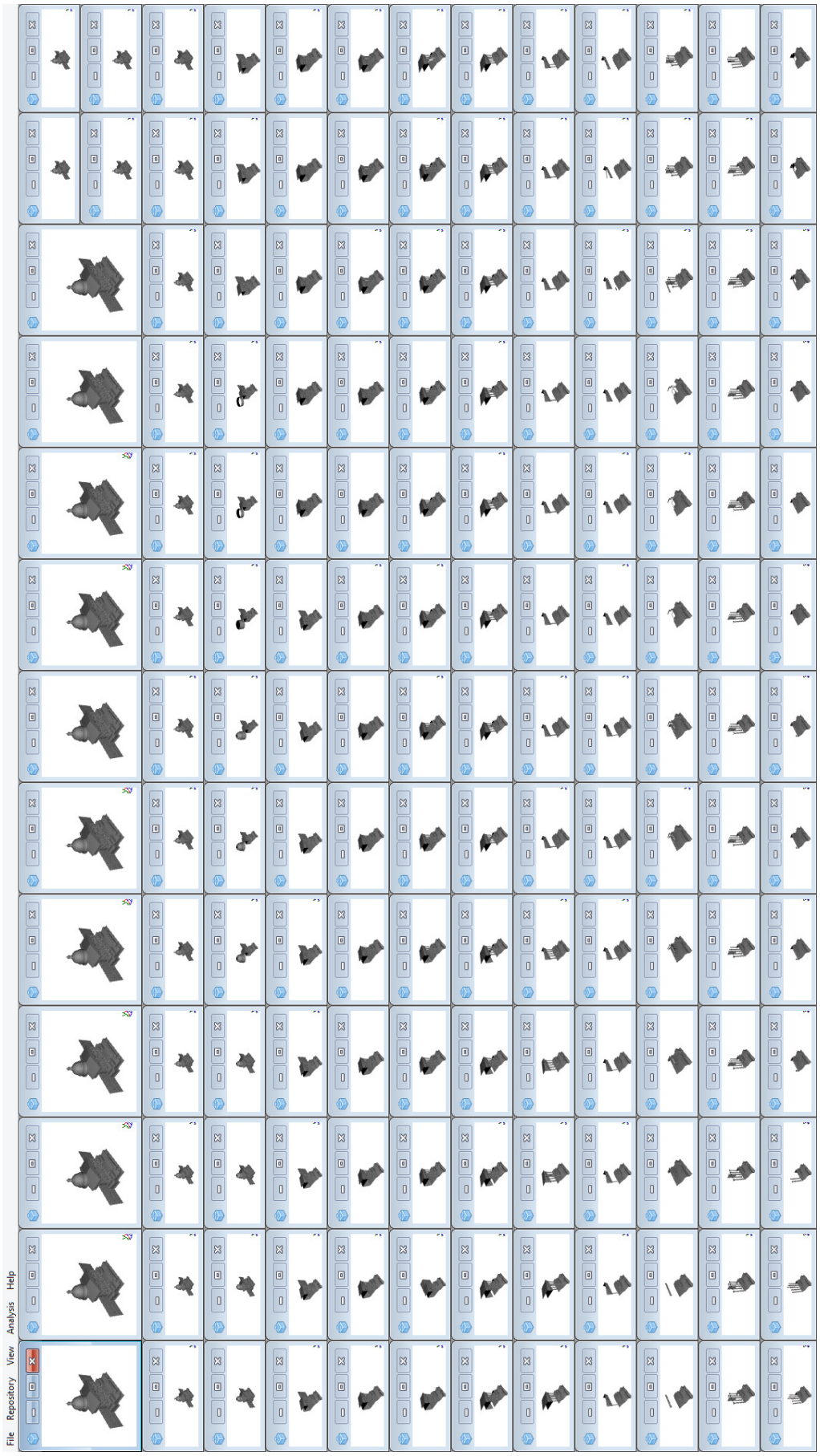


Figure E.6: *Portico dataset.*

Bibliography

[3DS12] 3DSys. 3d-diff, May 2012. Computer Software, URL: http://www.3dsys.fr/6.html.

[Ado08] Adobe. Iso 32000-1:2008 document management – portable document format – part 1: Pdf 1.7. Technical specification, International Organization for Standardization, July 2008. ISO 32000-1:2008.

[Adv11] Advanced Distributed Learning. 3d repository api documentation, September 2011. URL: https://github.com/adlnet/3D-Repository/wiki/API-Documentation.

[Adv13] Advanced Micro Devices, Inc. Amd delivers unified gaming strategy at gdc 2013, March 2013. Press Release.

[AFS06] Marco Attene, Bianca Falcidieno, and Michela Spagnuolo. Hierarchical mesh segmentation based on fitting primitives. *Vis. Comput.*, 22(3):181–193, March 2006.

[AG08] Renzo Angles and Claudio Gutierrez. Survey of graph database models. *ACM Comput. Surv.*, 40(1):1:1–1:39, February 2008.

[Age14] Danish Geodata Agency. Denmark in minecraft, April 2014. URL: http://eng.gst.dk/maps-topography/denmark-in-minecraft.

[Ala11] T. Alatalo. An entity-component model for extensible virtual worlds. *Internet Computing, IEEE*, 15(5):30–37, Sept 2011.

[ALS10] J. Chris Anderson, Jan Lehnardt, and Noah Slater. *CouchDB: The Definitive Guide*. O'Reilly Media, February 2010. ISBN-10: 0596155891.

[Ana96] Vera B. Anand. *Computer graphics and geometric modeling for engineers*. John Wiley & Sons, Inc., first edition, 1996. ISBN-10: 0471514179.

[App98] Apple, Inc. The webkit open source project, November 1998. Computer Software, URL: https://www.webkit.org.

[Aro14] Sameer Arora. Enabling the future of engineering. Whitepaper, Dassault Systèmes, S.A., August 2014.

[ASW09] Kerstin Altmanninger, Martina Seidl, and Manuel Wimmer. A survey on model versioning approaches. *International Journal of Web Information Systems*, 5(3):271–304, 2009.

[Au08] Wagner James Au. *The Making of Second Life: Notes from the New World*. Collins, first edition, February 2008. ISBN-10: 0061353205.

[Aus06] David Austerberry. *Digital asset management*. Focal Press, second edition, October 2006. ISBN-10: 0240808681.

[Aut03] Autodesk, Inc. Building information modeling. White paper, Autodesk Building Industry Solutions, San Rafael, CA, March 2003.

[Aut07] Autodesk, Inc. Navisworks, May 2007. Computer Software, URL: http://www.autodesk.com/products/navisworks/overview.

[Aut10] Autodesk, Inc. Multi-user collaboration with autodesk revit worksharing, October 2010. Whitepaper.

[Aut12] Autodesk, Inc. Autodesk screencast - a simple way to capture and share what you know, October 2012. Computer Software, URL: https://screencast.autodesk.com.

[Aut14a] Autodesk, Inc. AutoCAD 360, January 2014. Computer Software, URL: https://www.autocad360.com.

[Aut14b] Autodesk, Inc. BIM 360, January 2014. Computer Software, URL: http://www.autodesk.com/products/bim-360/overview.

[AW96] Tigran Andjelic and Michael Worboys. Version management for gis in a distributed environment. *Innovations in GIS*, 3:65–74, 1996.

[AYWM14] Sawsan AlHalawani, Yong-Liang Yang, Peter Wonka, and Niloy J. Mitra. What makes london work like london. *Computer Graphics Forum (Proceedings of SGP 2014)*, 33, 2014.

[Bau08] Petr Baudiš. Current concepts in version control systems. *Dobb's Journal*, page 46, 2008.

[BBB+10] Morten Bohøj, Nikolaj Gandrup Borchorst, Niels Olof Bouvin, Susanne Bødker, and Pär-Ola Zander. Timeline collaboration. In *Proc. SIGCHI*, pages 523–532, 2010.

[BBFG94] Steve Benford, John Bowers, Lennart E Fahlén, and Chris Greenhalgh. Managing mutual awareness in collaborative virtual environments. In *Proceedings of the conference on Virtual reality software and technology*, pages 223–236, 1994.

[BC08] Rich Bowen and Ken Coar. *Apache Cookbook: Solutions and Examples for Apache Administrators*. O'Reilly Media, second edition, January 2008. ISBN-10: 0596529945.

[BCC+05] Louis Bavoil, Steven P. Callahan, Patricia J. Crossno, Juliana Freire, and Huy T. Vo. Vistrails: Enabling interactive multiple-view visualizations. In *In IEEE Visualization 2005*, pages 135–142, 2005.

[BD94] Philip A. Bernstein and Umeshwar Dayal. An overview of repository technology. In *VLDB*, volume 94, pages 705–713, 1994.

[BEC+14] Eric Burns, David Easter, Rob Chadwick, David A Smith, and Carl Rosengrant. The virtual world framework: Collaborative virtual environments on the web. In *Virtual Reality (VR), 2014 iEEE*, pages 165–166. IEEE, 2014.

[BEJZ09] Johannes Behr, Peter Eschler, Yvonne Jung, and Michael Zöllner. X3DOM: a DOM-based HTML5/X3D integration model. In *Proceedings of the 14th International Conference on 3D Web Technology*, Web3D '09, pages 127–135, New York, NY, USA, 2009. ACM.

[Ben10] Marco Di Benedetto. SpiderGL. Visual Computing Laboratory - ISTI - CNR, January 2010. Computer Software, URL: http://spidergl.org.

[Ben11a] Bentley Systems, Inc. Assetwise: Infrastructure asset management software and services, September 2011. Computer Software, URL: http://www.bentley.com/en-GB/Products/assetwise.

[Ben11b] Bentley Systems, Inc. Projectwise information management and collaboration, September 2011. Computer Software, URL: http://www.bentley.com/en-GB/Products/projectwise+project+team+collaboration.

[BF08] Mark Barnes and Ellen Levy Finch. Collada - digital asset schema release 1.5.0. Technical specification, Khronos Group, April 2008.

[BFL+14] Robin Berjon, Steve Faulkner, Travis Leithead, Erika Doyle Navara, Edward O'Connor, Silvia Pfeiffer, and Ian Hickson. Html5 a vocabulary and associated apis for html and xhtml. Technical specification, World Wide Web Consortium, June 2014.

[BGT+10] Jing Bai, Shuming Gao, Weihua Tang, Yusheng Liu, and Song Guo. Design reuse oriented partial retrieval of cad models. *Computer-Aided Design*, 42(12):1069–1084, 2010.

[BHP99] Gill Barequet and Sariel Har-Peled. Efficiently approximating the minimum-volume bounding box of a point set in three dimensions. In *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '99, pages 82–91, Philadelphia, PA, USA, 1999. Society for Industrial and Applied Mathematics.

[BHS13] André R. Brodtkorb, Trond R. Hagen, and Martin L. Sætra. Graphics processing unit (gpu) programming strategies and trends in gpu computing. *Journal of Parallel and Distributed Computing*, 73(1):4–13, 2013.

[BJFS12] Johannes Behr, Yvonne Jung, Tobias Franke, and Timo Sturm. Using images and explicit binary container for efficient and incremental delivery of declarative 3d scenes on the web. In *Proceedings of the 17th International Conference on 3D Web Technology*, Web3D '12, pages 17–25, New York, NY, USA, 2012. ACM.

[BKFS13] David Birch, Paul H. J. Kelly, Anthony J. Field, and Alvise Simondetti. Computationally unifying urban masterplanning. In *Proceedings of the ACM International Conference on Computing Frontiers*, CF '13, pages 32:1–32:10, New York, NY, USA, 2013. ACM.

[BKK85] François Bancilhon, Won Kim, and Henry F. Korth. A model of cad transactions. In *Proceedings of the 11th International Conference on Very Large Data Bases - Volume 11*, VLDB '85, pages 25–33. VLDB Endowment, 1985.

[BKPS97] Thomas Ball, Jung-Min Kim, Adam A Porter, and Harvey P Siy. If your version control system could talk. In *ICSE Workshop on Process Modelling and Empirical Studies of Software Engineering*, 1997.

[BKS+05] Benjamin Bustos, Daniel A Keim, Dietmar Saupe, Tobias Schreck, and Dejan V Vranić. Feature-based similarity search in 3d object databases. *ACM Computing Surveys (CSUR)*, 37(4):345–387, 2005.

[Bla12] John M. Blain. *The Complete Guide to Blender Graphics: Computer Modeling and Animation*. A K Peters/CRC Press, first edition, May 2012. ISBN-10: 1466517034.

[Bla13] Sue Blackman. *Beginning 3D Game Development with Unity 4: All-In-One, Multi-Platform Game Development (Technology in Action)*. Apress, second edition, August 2013. ISBN-10: 1430248998.

[BLCR10] Diego Cordeiro Barboza, H Lima, Esteban Walter Gonzalez Clua, and Vinod EF Rebello. A simple architecture for digital games on demand using low performance resources under a cloud computing paradigm. In *Games and Digital Entertainment (SBGAMES), 2010 Brazilian Symposium on*, pages 33–39. IEEE, 2010.

[BLHL+01] Tim Berners-Lee, James Hendler, Ora Lassila, et al. The semantic web. *Scientific american*, 284(5):28–37, 2001.

[BM88] David Beech and Brom Mahbod. Generalized version control in an object-oriented database. In *Data Engineering, 1988. Proceedings. Fourth International Conference on*, pages 14–22. IEEE, 1988.

[BMC+96] Kurt Buehler, Lance McKee, Open GIS Consortium, et al. *The OpenGIS Guide: Introduction to Interoperable Geoprocessing: Part I of the Open Geodata Interoperability Specification (OGIS)*. Open GIS Consortium, Incorporated, 1996.

[BMZ+05] Jim Buckley, Tom Mens, Matthias Zenger, Awais Rashid, and Günter Kniesel. Towards a taxonomy of software change. *Journal of Software Maintenance and Evolution: Research and Practice*, 17(5):309–332, 2005.

[BO04] Don Burns and Robert Osfield. Open scene graph a: Introduction, b: Examples and applications. *Virtual Reality Conference, IEEE*, 0:265, 2004.

[BOSD+12] Elisa Bertino, Beng Chin Ooi, Ron Sacks-Davis, Kian-Lee Tan, Justin Zobel, Boris Shidlovsky, and Daniele Andronico. *Indexing Techniques for Advanced Database Systems*. Springer Publishing Company, Incorporated, 2012.

[Bou97] Thomas Boutell. Png (portable network graphics) specification version 1.0. *The Internet Engineering Task Force*, 1997.

[BPSM+08] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, Franois Yergeau, and John Cowan. Extensible markup language (xml) 1.0 (fifth edition). W3C recommendation, World Wide Web Consortium, November 2008.

[BRDA11] Rozenn Bouville Berthelot, Jérôme Royan, Thierry Duval, and Bruno Arnaldi. Scene graph adapter: an efficient architecture to improve interoperability between 3d formats and 3d applications engines. In *Proceedings of the 16th International Conference on 3D Web Technology*, Web3D '11, pages 21–29, New York, NY, USA, 2011. ACM.

[Bro87] Frederick P. Brooks, Jr. Walkthrough—a dynamic graphics system for simulating virtual buildings. In *Proceedings of the 1986 Workshop on Interactive 3D Graphics*, I3D '86, pages 9–21, New York, NY, USA, 1987. ACM.

[Bro96] John Brooke. SUS: A quick and dirty usability scale. In P. W. Jordan, B. Weerdmeester, A. Thomas, and I. L. Mclelland, editors, *Usability evaluation in industry*. CRC Press, London, 1996.

[Bro14] Jim Brown. Issue in focus: Integrating product design and development environments battling complexity, improving productivity, and compressing time. Whitepaper, Tech Clarity, Inc., 2014.

[Bru09] Paul Brunt. Glge webgl library/framework, December 2009. Computer Software, URL: http://www.glge.org.

[BS00] Uwe M Borghoff and Johann H Schlichter. *Computer-supported cooperative work*. Springer, 2000. ISBN: 978-3-642-08631-1.

[BS07] Emil Brink and Eskil Steenberg. Uni-verse project, September 2007. Computer Software, URL: http://www.quelsolaar.com/verse/index.html.

[BS08] Jasmin Blanchette and Mark Summerfield. *C++ GUI Programming with Qt4*. Prentice Hall, second edition, February 2008. ISBN-10: 0132354160.

[BSF83] C. L. Blackburn, O. O. Storaasli, and R. E. Fulton. The role and application of data base management in integrated computer-aided design. *Journal of Aircraft*, 20(8):717–725, 1983.

[BSV98] Michael Berger, Alexander Schill, and Gerd Völksen. Supporting autonomous work and reintegration in collaborative systems. In *Coordination Technology for Collaborative Applications*, pages 177–198. Springer, 1998.

[bui13] buildingSmart. Industry foundation classes (ifc) for data sharing in the construction and facility management industries. Technical specification, buildingSMART International Ltd, July 2013. ISO 16739:2013.

[Bur05] Ed Burnette. *Eclipse IDE Pocket Guide*. O'Reilly Media, first edition, August 2005. ISBN-10: 0596100655.

[Cab10] Ricardo Cabello. three.js - javascript 3d library, April 2010. Computer Software, URL: http://threejs.org.

[Cal09] Steven Callahan. Provenance explorer for maya 2008/2009. Technical report, VisTrails, Inc., May 2009.

[Cap12] Capvidia, Inc. Comparevidia validates translations of cad models ensuring data integrity, May 2012. URL: http://www.capvidia.com/capvidia-products/comparevidia-cad-model-translation-validation.

[Cat11] Rick Cattell. Scalable sql and nosql data stores. *ACM SIGMOD Record*, 39(4):12–27, 2011.

[CB04] Thomas Connolly and Carolyn Begg. *Database Systems: A Practical Approach to Design, Implementation and Management*. Addison Wesley, 2004. ISBN: 978-0321210258.

[CBM97] Rikk Carey, Gavin Bell, and Chris Marrin. Information technology – computer graphics and image processing – the virtual reality modeling language – part 1: Functional specification and utf-8 encoding. Iso standard, The VRML Consortium Incorporated, January 1997. ISO/IEC 14772-1:1997.

[CCLT+10] Justin Clark-Casey, Crista Lopes, Melanie Thielker, Michael Cerquoni, and Ben Esplin. Open simulator wiki – compatible viewers. Overte Foundation, December 2010. URL: http://opensimulator.org/wiki/Connecting.

[CCT+11] Kuan-Ta Chen, Yu-Chun Chang, Po-Han Tseng, Chun-Ying Huang, and Chin-Laung Lei. Measuring the latency of cloud gaming systems. In *Proceedings of the 19th ACM international conference on Multimedia*, pages 1269–1272. ACM, 2011.

[Cel12] Joe Celko. *Joe Celko's Trees and Hierarchies in SQL for Smarties (The Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann, December 2012. ISBN-10: 0123877334.

[CFGS12] Mark Claypool, David Finkel, Alexander Grant, and Michael Solano. Thin to win? network performance analysis of the onlive thin client game system. In *Network and Systems Support for Games (NetGames), 2012 11th Annual Workshop on*, pages 1–6. IEEE, 2012.

[CGF09] Xiaobai Chen, Aleksey Golovinskiy, and Thomas Funkhouser. A benchmark for 3D mesh segmentation. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 28(3), August 2009.

[CGW+14] Hsiang-Ting Chen, Tovi Grossman, Li-Yi Wei, Ryan M. Schmidt, Björn Hartmann, George Fitzmaurice, and Maneesh Agrawala. History assisted view authoring for 3d models. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 2027–2036, New York, NY, USA, 2014. ACM.

[CH93] Christer Carlsson and Olof Hagsand. Dive-a platform for multi-user virtual environments. *Computers & graphics*, 17(6):663–669, 1993.

[Cha09] Scott Chacon. *Pro Git*. Apress, first edition, 2009. ISBN: 1430218339.

[Chi09] Bill Childers. Run your own virtual reality with opensim. *Linux Journal*, 2009(179):6, 2009.

[Cho11] Kristina Chodorow. *Scaling MongoDB*. O'Reilly Media, first edition, March 2011. ISBN-10: 1449303218.

[Chu13] Won Chun. webgl-loader - simple, fast, and compact mesh compression for webgl. Google, Inc., 2013. Computer Software, URL: http://code.google.com/p/webgl-loader.

[CK86] Hong-Tai Chou and Won Kim. A unifying framework for version control in a cad environment. In *Proceedings of the 12th International Conference on Very Large Data Bases*, VLDB '86, pages 336–344, San Francisco, CA, USA, 1986. Morgan Kaufmann Publishers Inc.

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition*. McGraw-Hill Science/Engineering/Math, second edition, July 2001. ISBN-10: 0070131511.

[CO00] David Sánchez Crespillo and Antonio Francisco Bennasar Obrador. Implementation of mechanisms for concurrent 3d design and visualization. In *Proceedings of the International Conference on Information Visualisation*, page 179. IEEE Computer Society, 2000.

[Coa09] Steve Coast. OpenStreetMap, 2009. Computer Software, URL: http://www.openstreetmap.org.

[Com79] Douglas Comer. Ubiquitous b-tree. *ACM Computing Surveys (CSUR)*, 11(2):121–137, 1979.

[CRLR13] David Catuhe, David Rousset, Pierre Lagarde, and Michel Rousseau. Babylon.js - webgl. simple. powerful., July 2013. Computer Software, URL: http://www.babylonjs.com.

[CRS+13] Senthil K. Chandrasegaran, Karthik Ramani, Ram D. Sriram, Imré Horváth, Alain Bernard, Ramy F. Harik, and Wei Gao. The evolution, challenges, and future of knowledge representation in product design systems. *Computer-Aided Design*, 45(2):204–228, 2013.

[CS98] Elizabeth F. Churchill and Dave Snowdon. Collaborative virtual environments: an introductory review of issues and systems. *Virtual Reality*, 3(1):3–15, 1998.

[CWC11] Hsiang-Ting Chen, Li-Yi Wei, and Chun-Fa Chang. Nonlinear revision control for images. *ACM Trans. Graph.*, 30(4):105:1–105:10, Aug 2011.

[CWSR12] Sharon Choy, Bernard Wong, Gwendal Simon, and Catherine Rosenberg. The brewing storm in cloud gaming: A measurement study on cloud to end-user latency. In *Proceedings of the 11th annual workshop on network and systems support for games*, page 2. IEEE Press, 2012.

[DA10] Petros Daras and Apostolos Axenopoulos. A 3d shape retrieval framework supporting multimodal queries. *International Journal of Computer Vision*, 89(2-3):229–247, 2010.

[Das11] Dassault Systèmes, S.A. Catia, September 2011. Computer Software, URL: http://www.3ds.com/products-services/catia.

[Das12a] Dassault Systèmes, S.A. 3dvia shape for maps, August 2012. Computer Software, URL: http://www.3dvia.com/products/3dvia-shape-for-maps.

[Das12b] Dassault Systèmes, S.A. Enovia, December 2012. Computer Software, URL: http://www.3ds.com/products/enovia/solutions.

[DD13] Randi L. Derakhshani and Dariush Derakhshani. *Autodesk 3ds Max 2014 Essentials: Autodesk Official Press*. John Wiley & Sons, first edition, June 2013. ISBN-10: 1118575148.

[DDB11] Klaus R. Dittrich, Umeshwar Dayal, and Alejandro P. Buchmann. *On Object-Oriented Database Systems*. Springer Publishing Company, Incorporated, 1st edition, 2011.

[Dep05] Deputy Prime Minister Office. Explanatory memorandum to the town and country planning (major infrastructure project inquiries procedure) (England) rules 2005, the office for the deputy prime minister and the department for constitutional affairs, July 2005.

[Deu96a] L. Peter Deutsch. Deflate compressed data format specification version 1.3. *The Internet Engineering Task Force*, May 1996.

[Deu96b] L. Peter Deutsch. Gzip file format specification version 4.3. *The Internet Engineering Task Force*, May 1996.

[DF08] Susan B. Davidson and Juliana Freire. Provenance and scientific workflows: challenges and opportunities. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 1345–1350, New York, NY, USA, 2008. ACM.

[DG08] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51:107–113, January 2008.

[DGG+13] Elif Dede, Madhusudhan Govindaraju, Daniel Gunter, Richard Shane Canon, and Lavanya Ramakrishnan. Performance evaluation of a mongodb and hadoop platform for scientific data analysis. In *Proceedings of the 4th ACM Workshop on Scientific Cloud Computing*, Science Cloud '13, pages 13–20, New York, NY, USA, 2013. ACM.

[DKP11] Jonathan D. Denning, William B. Kerr, and Fabio Pellacini. Meshflow: interactive visualization of mesh construction sequences. *ACM Trans. Graph.*, 30(4):66:1–66:8, July 2011.

[DLFPT09] Andrea De Lucia, Rita Francese, Ignazio Passero, and Genoveffa Tortora. Development and evaluation of a virtual campus on second life: The case of seconddmi. *Computers & Education*, 52(1):220–233, 2009.

[DMBP04] Aaron CE Darling, Bob Mau, Frederick R Blattner, and Nicole T Perna. Mauve: multiple alignment of conserved genomic sequence with rearrangements. *Genome research*, 14(7):1394–1403, 2004.

[DMS14] Jozef Doboš, Niloy J. Mitra, and Anthony Steed. 3d timeline: Reverse engineering of a part-based provenance from consecutive 3d models. In *Computer Graphics Forum*, volume 33, pages 135–144. Wiley Online Library, 2014.

[Dou95] Paul Dourish. The parting of the ways: divergence, data management and collaborative work. In *Proceedings of the fourth conference on European Conference on Computer-Supported Cooperative Work*, pages 215–230, Norwell, MA, USA, 1995. Kluwer Academic Publishers.

[DP13] Jonathan D. Denning and Fabio Pellacini. Meshgit: Diffing and merging meshes for polygonal modeling. *ACM Trans. Graph.*, 32(4):35:1–35:10, July 2013.

[DR12] Kaustuv DeBiswas and Nitin Rao. Sunglass, January 2012. Computer Software, URL: http://sunglass.io.

[DS10] Darius Dadgari and Wolfgang Stuerzlinger. Novel user interfaces for diagram versioning and differencing. In *Proceedings of the 24th BCS Interaction Specialist Group Conference*, BCS '10, pages 62–71, Swinton, UK, UK, 2010. British Computer Society.

[DS12a] Jozef Doboš and Anthony Steed. 3D Diff: an interactive approach to mesh differencing and conflict resolution. In *SIGGRAPH Asia 2012 Technical Briefs*, SA '12, pages 20:1–20:4, New York, NY, USA, 2012. ACM.

[DS12b] Jozef Doboš and Anthony Steed. 3D revision control framework. In *Proceedings of the 17th International Conference on 3D Web Technology*, Web3D '12, pages 121–129, New York, NY, USA, 2012. ACM.

[DSMBMM+11] David Dominguez-Sal, Norbert Martinez-Bazan, Victor Muntes-Mulero, Pere Baleta, and Josep Lluis Larriba-Pey. A discussion on the design of graph database benchmarks. In *Performance Evaluation, Measurement and Characterization of Complex Systems*, pages 25–40. Springer, 2011.

[DSR+13] Jozef Doboš, Kristian Sons, Dmitri Rubinstein, Philipp Slusallek, and Anthony Steed. Xml3drepo: a rest api for version controlled 3d assets on the web. In *Proceedings of the 18th International Conference on 3D Web Technology*, Web3D '13, pages 47–55, New York, NY, USA, 2013. ACM.

[DSS12] Jozef Doboš, Alvise Simondetti, and Anthony Steed. Visualizing 3d models in aid of public consultation. In *SIGGRAPH Asia 2012 Symposium on Apps*, SA '12, pages 9:1–9:1, New York, NY, USA, 2012. ACM.

[Dun11] Sean C. Duncan. Minecraft, beyond construction and survival. *Well Played: a journal on video games, value and meaning*, 1(1):1–22, 2011.

[DW06] Stijn Dekeyser and Richard Watson. Extending google docs to collaborate on research papers. Technical report, The University of Southern Queensland, Australia, 2006.

[Eas99] Charles M. Eastman. *Building product models: computer environments, supporting design and construction*. CRC press, 1999.

[Eas12] Chuck Eastman. The evolution of aec interoperability. European Group for Intelligent Computing in Engineering, July 2012.

[ECM13] ECMA. Ecma-404 the json data interchange standard. Technical specification, Ecma International, October 2013.

[EFL+74] Charles M. Eastman, David Fisher, Gilles Laufe, Joseph Lividini, Douglas Stoker, and Christos Yessios. An outline of the building description system. research report no. 50. *Education Resources Information Center*, September 1974.

[ENT11] Mark E. Easterfield, Richard G. Newell, and David G. Theriault. Smallworld technical paper no. 4 - version management in gis - applications and techniques, August 2011.

[ES11] Koos Eissen and Roselein Steur. *Sketching: The Basics*. BIS Publishers B.V., 2011. ISBN-10: 9063692536.

[ESR99] ESRI. Arcgis, December 1999. Computer Software, URL: http://www.esri.com/software/arcgis.

[FF14] Obie Fernandez and Kevin Faustino. *The Rails 4 Way*. Addison-Wesley Professional, third edition, June 2014. ISBN-10: 0321944275.

[FGLW08] Rui Fang, Afzal Godil, Xiaolan Li, and Asim Wagan. A new shape benchmark for 3d object retrieval. In *Advances in Visual Computing*, pages 381–392. Springer, 2008.

[FGM+99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1, 1999.

[FHK06] Sally Floyd, Mark Handley, and Eddie Kohler. Datagram congestion control protocol (dccp). *The Internet Engineering Task Force*, March 2006.

[Fie00] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000. AAI9980887.

[FJJ00] Jon Ferraiolo, Fujisawa Jun, and Dean Jackson. *Scalable vector graphics (SVG) 1.0 specification*. iuniverse, 2000.

[FK00] Tom G. Farr and Mike Kobrick. Shuttle radar topography mission produces a wealth of data. *Eos, Transactions American Geophysical Union*, 81(48):583–585, 2000.

[FK05] N. Freed and J. Klensin. Media type specifications and registration procedures. Technical report, The Internet Society, December 2005.

[Fla11] David Flanagan. *JavaScript: The Definitive Guide*. O'Reilly Media, sixth edition, May 2011. ISBN-10: 0596805527.

[Fou10] Open Wonderland Foundation. Project wonderland basic features, November 2010. Computer Software, URL: http://openwonderland.org/about/features.

[FR13] Stewart Fotheringham and Peter Rogerson. *Spatial analysis and GIS*. CRC Press, 2013.

[FW07] Sabrina Förtsch and Bernhard Westfechtel. Differencing and merging of software diagrams - state of the art and challenges. In *ICSOFT (SE)*, pages 90–99, 2007.

[Gai12] Gaikai, Inc. Gaikai open cloud gaming platform, January 2012. Computer Software, URL: http://www.gaikai.com.

[GAL+09] Floraine Grabler, Maneesh Agrawala, Wilmot Li, Mira Dontcheva, and Takeo Igarashi. Generating photo manipulation tutorials by demonstration. *ACM SIGGRAPH*, 28(3):66:1–66:9, 2009.

[Gar13] Juan Jiménez García. *Building 3D Models with modo 701*. Packt Publishing, first edition, October 2013. ISBN-10: 1849692467.

[GAW+11] Michael Gleicher, Danielle Albers, Rick Walker, Ilir Jusufi, Charles D. Hansen, and Jonathan C. Roberts. Visual comparison for information visualization. *Information Visualization*, 10(4), oct 2011.

[GB95] Chris Greenhalgh and Steve Benford. Massive: a distributed virtual reality system incorporating spatial trading. In *Distributed Computing Systems, 1995., Proceedings of the 15th International Conference on*, pages 27–34. IEEE, 1995.

[GCBA12] Aaron Griffith, Andrew Chin, Andrew Brown, and Alejandro Aguilera. Minecraft overviewer, March 2012. Computer Software. URL: http://overviewer.org.

[GE06] Rich Gibson and Schuyler Erle. *Google Maps Hacks*. O'Reilly Media, January 2006.

[Gee09] Marcus Geelnard. OpenCTM format, December 2009. Software Library.

[GF08] Aleksey Golovinskiy and Thomas Funkhouser. Randomized cuts for 3d mesh analysis. *ACM Trans. Graph.*, 27(5):145:1–145:12, December 2008.

[GF09] Aleksey Golovinskiy and Thomas Funkhouser. Consistent segmentation of 3D models. *Computers and Graphics (Shape Modeling International 09)*, 33(3):262–269, June 2009.

[Gle98] Wolf L. Glende. The boeing 777: a look back. In *AGARD CONFERENCE PROCEEDINGS AGARD CP*, pages 5–5. AGARD, 1998.

[GM08] Hector Garcia-Molina. *Database systems: the complete book*. Pearson Education India, 2008.

[GMF10] Tovi Grossman, Justin Matejka, and George Fitzmaurice. Chronicle: Capture, exploration, and playback of document workflow histories. In *User Interface Software and Technology Symposium*, pages 143–152. ACM, 2010.

[GN14] Marko Gargenta and Masumi Nakamura. *Learning Android: Develop Mobile Apps Using Java and Eclipse*. O'Reilly Media, second edition, January 2014. ISBN-10: 1449319238.

[Goo08a] Google, Inc. Google map maker, June 2008. Computer Software, URL: http://www.google.com/mapmaker.

[Goo08b] Google, Inc. V8 javascript engine, July 2008. Computer Software.

[Goo12] Google, Inc. Google drive, January 2012. Computer Software, URL: http://drive.google.com.

[Goo13] Google, Inc. Blink, April 2013. Computer Software.

[Goo14a] Google, Inc. Chrome remote desktop, August 2014. Computer Software, URL: https://chrome.google.com/webstore/detail/chrome-remote-desktop.

[Goo14b] Google, Inc. Protocol buffers encoding, September 2014. Technical Specification.

[Gra09] SE Graphisoft. Bim server, September 2009. Computer Software, URL: https://www.graphisoft.com/bim-server.

[Gro11] BIM Industry Working Group. Government construction strategy. Policy paper, Cabinet Office, May 2011.

[GS99] T. Goddard and V. S. Sunderam. Toolspace: Web based 3d collaboration. In *Proceedings of the Fourth Symposium on Virtual Reality Modeling Language*, VRML '99, pages 161–166, New York, NY, USA, 1999. ACM.

[GS10] Jason van Gumster and Robert Shimonski. *GIMP Bible*. John Wiley & Sons, first edition, February 2010. ISBN-10: 0470523972.

[GSM+08] Maurizio Gibin, Alex Singleton, Richard Milton, Pablo Mateos, and Paul Longley. An exploratory cartographic visualisation of london through the google maps api. *Applied Spatial Analysis and Policy*, 1:85–97, 2008.

[GWH01] Michael Garland, Andrew Willmott, and Paul S. Heckbert. Hierarchical face clustering on polygonal surfaces. In *I3D*, 2001.

[GZ12] Marcus Goetz and Alexander Zipf. Towards defining a framework for the automatic derivation of 3d citygml models from volunteered geographic information. *International Journal of 3-D Information Modeling (IJ3DIM)*, 1(2):1–16, 2012.

[HÏ13] Urs Hölzle. A second spring of cleaning. Google, Inc., March 2013. URL: http://googleblog.blogspot.co.uk/2013/03/a-second-spring-of-cleaning.html.

[Haz08] Dan Hazel. Using rational numbers to key nested sets. *CoRR*, abs/0806.3115, 2008.

[HB05] Ian Howell and Bob Batcheler. Building information modeling two years later–huge potential, some success and several limitations. *The Laiserin Letter*, 22, 2005.

[Hen08] Mark Hendrickson. Otoy developing server-side 3d rendering technology, July 2008. URL: http://techcrunch.com/2008/07/09/otoy-developing-server-side-3d-rendering-technology.

[Hew06] Joe Hewitt. Firebug, January 2006. Computer Software, URL: http://getfirebug.com.

[HFL12] Ruizhen Hu, Lubin Fan, and Ligang Liu. Co-segmentation of 3d shapes via subspace clustering. In *Computer graphics forum*, volume 31, pages 1703–1713. Wiley Online Library, 2012.

[HGF+02] Daniel M. Hellerstein, Yaron Y. Goland, Anja Feldmann, Jeffrey C. Mogul, Balachander Krishnamurthy, Fred Douglis, and Arthur van Hoff. Delta encoding in http. *Delta*, 2002.

[HHCC13] Chun-Ying Huang, Cheng-Hsin Hsu, Yu-Chun Chang, and Kuan-Ta Chen. Gaminganywhere: An open cloud gaming system. In *Proceedings of the 4th ACM multimedia systems conference*, pages 36–47. ACM, 2013.

[HHLD11] Jing Han, E Haihong, Guan Le, and Jian Du. Survey on nosql database. In *Pervasive computing and applications (ICPCA), 2011 6th international conference on*, pages 363–366. IEEE, 2011.

[HKG11] Qixing Huang, Vladlen Koltun, and Leonidas Guibas. Joint shape segmentation with linear programming. *ACM Transactions on Graphics (TOG)*, 30(6), December 2011.

[HM76] James Wayne Hunt and M. Douglas McIlroy. An algorithm for differential file comparison. Technical Report 41, AT&T Bell Laboratories, 1976.

[HM04] David Heesom and Lamine Mahdjoubi. Trends of 4d cad applications for construction planning. *Construction Management and Economics*, 22(2):171–182, 2004.

[HMRL95] Yasuaki Honda, Kouichi Matsuda, Jun Rekimoto, and Rodger Lea. Virtual society: Extending the www to support a multi-user interactive shared 3d environment. In *Proceedings of the First Symposium on Virtual Reality Modeling Language*, VRML '95, pages 109–116, New York, NY, USA, 1995. ACM.

[Hní10] Jiří Hnídek. Introduction of new verse protocol. In *9th International Blender Conference*, Amsterdam, Netherlands, November 2010.

[Hní11] Jiří Hnídek. Network protocols for applications of shared virtual reality. In *Communications proceedings of 19th International Conference on Computer Graphics*, pages 31–38, Plzeň, Czech Republic, 2011. Visualization and Computer Vision.

[Hní12] Jiří Hnídek. Draft of verse protocol, November 2012. Technical Specification, URL: https://github.com/verse/verse/wiki/Specification.

[Hní13] Jiří Hnídek. Verse mongodb backend, November 2013. Technical Specification, URL: https://github.com/verse/verse/wiki/MongoDB-Backend.

[Hol10] Sean Hollister. Onlive game system review. *Engadget*, December 2010. URL: http://www.engadget.com/2010/12/16/onlive-game-system-review.

[Hol12] Sean Hollister. Gaikai enters closed beta, we get an exclusive first look, December 2012. URL: http://www.engadget.com/2010/12/02/gaikai-enters-closed-beta-we-get-an-exclusive-first-look.

[Hop96] Hugues Hoppe. Progressive meshes. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '96, pages 99–108, New York, NY, USA, 1996. ACM.

[HS99] Gísli R Hjaltason and Hanan Samet. Distance browsing in spatial databases. *ACM Transactions on Database Systems (TODS)*, 24(2):265–318, 1999.

[HS02] Andrew Hudson-Smith. 30 days in active worlds: community, design and terrorism in a virtual world. In *The social life of avatars*, pages 77–89. Springer, 2002.

[HSFP99] Gerd Hesina, Dieter Schmalstieg, Anton Furhmann, and Werner Purgathofer. Distributed open inventor: a practical approach to distributed 3d graphics. In *Proceedings of the ACM symposium on Virtual reality software and technology*, VRST '99, pages 74–81, New York, NY, USA, 1999. ACM.

[HVT98] James Wayne Hunt, Kiem-Phong Vo, and Walter F. Tichy. Delta algorithms: an empirical analysis. *ACM Transactions on Software Engineering and Methodoly (TOSEM)*, 7:192–214, Apr 1998.

[HvW08] Danny Holten and Jarke J. van Wijk. Visual comparison of hierarchically organized data. *Comput. Graph. Forum*, 27(3), 2008.

[HXM+13] Shi-Min Hu, Kun Xu, Li-Qian Ma, Bin Liu, Bi-Ye Jiang, and Jue Wang. Inverse image editing: Recovering a semantic editing history from a before-and-after image pair. *TOG*, 32(6), 2013.

[Ihr13] Colin Ihrig. *Pro Node.js for Developers*. APRESS, November 2013. ISBN-10: 1430258608.

[Inc04] Wavefront Technologies Inc. Object files, June 2004. Technical Specification.

[Ior10] Borislav Iordanov. Hypergraphdb: a generalized graph database. In *Web-Age Information Management*, pages 25–36. Springer, 2010.

[JIS+13] Joint Technical Committee ISO/IEC JTC 1, Information technology, Subcommittee 24, Computer graphics, image processing and environmental data representation, and Web3D Consortium, Inc. X3d architecture and base components v3. Iso standard, Web3D Consortium, Inc., November 2013. ISO/IEC IS 19775-1:2013.

[JKIR06] Subramaniam Jayanti, Yagnanarayanan Kalyanaraman, Natraj Iyer, and Karthik Ramani. Developing an engineering shape benchmark for cad models. *Computer-Aided Design*, 38(9):939–953, 2006.

[JL94] Daniel Jackson and David A Ladd. Semantic diff: A tool for summarizing the effects of modifications. In *Software Maintenance, 1994. Proceedings., International Conference on*, pages 243–252. IEEE, 1994.

[JLH+13] Yvonne Jung, Max Limper, Pasquale Herzig, Karsten Schwenk, and Johannes Behr. Fast and efficient vertex data representations for the web. In *Proceedings of the International Conference on Computer Graphics Theory*, GRAPP '13, pages 601–606, 2013.

[Joc12] Rainer Jochem. Openstreetmap 3d viewer and tools, September 2012. URL: http://xml3d.org/2012/09/openstreetmap-3d-viewer-and-tools.

[Joh88] Robert Johansen. *GroupWare: Computer Support for Business Teams*. The Free Press, New York, NY, USA, 1988.

[JSE05] Jens Jacobsen, Tilman Schlenker, and Lisa Edwards. *Implementing a digital asset management system: for animation, computer games, and web development*. Elsevier Focal Press, 2005.

[JTMT09] Leslie Jarmon, Tomoko Traphagan, Michael Mayrath, and Avani Trivedi. Virtual world teaching, experiential learning, and assessment: An interdisciplinary communication course in second life. *Computers & Education*, 53(1):169–182, 2009.

[JTRS12] Arjun Jain, Thorsten Thormählen, Tobias Ritschel, and Hans-Peter Seidel. Exploring shape variations by 3d-model decomposition and part-based recombination. *Comp. Graph. Forum (Proc. Eurographics 2012)*, 31(2), 2012.

[Jun10] Gregory Junker. *Pro OGRE 3D Programming (Expert's Voice in Open Source)*. Springer, first edition, September 2010. ISBN-10: 1590597109.

[KASS14] Anne van Kesteren, Julian Aubourg, Jungkee Song, and Hallvord R. M. Steen. XMLHttpRequest level 1, January 2014. W3C Working Draft 30.

[Kat90] Randy H. Katz. Toward a unified framework for version modeling in engineering databases. *ACM Computing Surveys (CSUR)*, 22(4):375–409, 1990.

[Kay10] Lindsay Kay. Scenejs, August 2010.

[Kaz93] Rick Kazman. Making waves: On the design of architectures for low-end distributed virtual environments. In *Virtual Reality Annual International Symposium, 1993., 1993 IEEE*, pages 443–449. IEEE, 1993.

[KBB+13] Artiom Kovnatsky, Michael M Bronstein, Alexander M Bronstein, Klaus Glashoff, and Ron Kimmel. Coupled quasi-harmonic bases. In *Computer Graphics Forum*, volume 32, pages 439–448. Wiley Online Library, 2013.

[Kel03] Tom Kelly. Scalable tcp: Improving performance in highspeed wide area networks. *ACM SIGCOMM Computer Communication Review*, 33(2):83–91, 2003.

[KF92] David Kurlander and Steven Feiner. A history-based macro by example system. In *Proceedings of the 5th annual ACM symposium on User interface software and technology*, UIST '92, pages 99–106, New York, NY, USA, 1992. ACM.

[KFH10] David Koller, Bernard Frischer, and Greg Humphreys. Research challenges for digital archives of 3d cultural heritage models. *J. Comput. Cult. Herit.*, 2:7:1–7:17, January 2010.

[KGB11] Ravikanth V. Kothuri, Albert Godfrind, and Euro Beinat. *Pro Oracle Spatial for Oracle Database 11g*. Apress, first edition, December 2011. ISBN-10: 1430242876.

[Khr13] Khronos Group. Typed array specification. Technical specification, editor's draft, Khronos Working Draft, July 2013.

[KHS10] Evangelos Kalogerakis, Aaron Hertzmann, and Karan Singh. Learning 3d mesh segmentation and labeling. *ACM Transactions on Graphics (TOG)*, 29(4):102, 2010.

[Kil96] Mark J. Kilgard. *OpenGL programming for the X Window System*. Addison Wesley Longman Publishing Co., Inc., 1996.

[KJS07] Vladislav Kreavoy, Dan Julius, and Alla Sheffer. Model composition from interchangeable components. In *PG '07*, pages 129–138, 2007.

[KKM+03] H-P. Kriegel, Peer Kroger, Zahi Mashael, Martin Pfeifle, Marcö Potke, and Thomas Seidl. Effective similarity search on voxelized cad objects. In *Database Systems for Advanced Applications, 2003.(DASFAA 2003). Proceedings. Eighth International Conference on*, pages 27–36. IEEE, 2003.

[KL84] Randy H. Katz and Tobin J. Lehman. Database support for versions and alternatives of large design files. *Software Engineering, IEEE Transactions on*, SE-10(2):191–200, March 1984.

[KLM+13] Vladimir G. Kim, Wilmot Li, Niloy J. Mitra, Siddhartha Chaudhuri, Stephen DiVerdi, and Thomas Funkhouser. Learning part-based templates from large collections of 3d shapes. *ACM Transactions on Graphics (TOG)*, 32(4):70, 2013.

[Klo06] Jane E. Klobas. *Wikis: Tools for Information Work and Collaboration (Information Professional)*. Chandos Publishing (Oxford) Ltd, June 2006. ISBN-10: 1843341786.

[KOL+09] Lindsay Kay, Andreas Osowski, Rehno Lindeque, Salomon Brys, and Stephen Bannasch. Scenejs, 2009. Computer Software, URL: http://scenejs.org.

[Kor10] Maria Korolov. Laid off wonderland developers to continue project. *Hypergrid Business*, February 2010. URL: http://www.hypergridbusiness.com/2010/02/laid-off-wonderland-developers-to-continue-project.

[KSR+12] Felix Klein, Kristian Sons, Dmitri Rubinstein, Sergiy Byelozyorov, Stefan John, and Philipp Slusallek. Xflow - Declarative Data Processing for the Web. In *Proceedings of the 17th International Conference on Web 3D Technology*, Web3D '12, pages 37–45, Los Angeles, California, 2012. ACM.

[KTA+11] Ranjitha Kumar, Jerry O. Talton, Salman Ahmad, Tim Roughgarden, and Scott R. Klemmer. Flexible tree matching. In Toby Walsh, editor, *IJCAI*, pages 2674–2679. IJCAI/AAAI, 2011.

[Kuh10] Ron Kuhfeld. Bentley's integrated structural modeling brings structural engineers into integrated project workflows. Bentley Systems, Inc., January 2010.

[KVMP12] Panayiotis Koutsabasis, Spyros Vosinakis, Katerina Malisova, and Nikos Paparounas. On the value of virtual worlds for collaborative design. *Design Studies*, 33(4):357–390, 2012.

[KY11] Jonathan Kaplan and Nicole Yankelovich. Open wonderland: an extensible virtual world architecture. *Internet Computing, IEEE*, 15(5):38–45, 2011.

[LAM76] John C. Thomas Lance A. Miller. Behavioral issues in the use of interactive systems. Technical report, IBM Research Laboratory, December 1976.

[LDSS99] Aaron WF Lee, David Dobkin, Wim Sweldens, and Peter Schröder. Multiresolution mesh morphing. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 343–350. ACM Press/Addison-Wesley Publishing Co., 1999.

[Len08] Mark Lentczner. Second life grid open grid protocol, December 2008. URL: http://wiki.secondlife.com/wiki/SLGOGP_Draft_1.

[LHKR12] Shuvendu Lahiri, Chris Hawblitzel, Ming Kawaguchi, and Henrique Rebelo. Symdiff: A language-agnostic semantic diff tool for imperative programs. In *Computer Aided Verification (CAV '12) (Tool description)*. Springer, July 2012.

[LHM97] Rodger Lea, Yasuaki Honda, and Kouichi Matsuda. Virtual society: Collaboration in 3d spaces on the internet. *Computer Supported Cooperative Work (CSCW)*, 6(2-3):227–250, 1997.

[Lin10] Linden Research, Inc. Second life – system requirements, September 2010. URL: http://secondlife.com/support/system-requirements/?lang=en-US.

[Lin11] Linden Research, Inc. Second life mesh, August 2011. URL: http://wiki.secondlife.com/wiki/Mesh.

[Liu99] Mengchi Liu. On cad databases. In *Electrical and Computer Engineering, 1999 IEEE Canadian Conference on*, volume 1, pages 325–330, May 1999.

[LJBA13] Max Limper, Yvonne Jung, Johannes Behr, and Marc Alexa. The pop buffer: Rapid progressive clustering by geometry quantization. *Computer Graphics Forum*, 32(7):197–206, 2013.

[LP02] Peter Lindstrom and Valerio Pascucci. Terrain simplification simplified: A general framework for view-dependent out-of-core visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 8(3):239–254, 2002.

[LTBF14] Max Limper, Maik Thöner, Johannes Behr, and Dieter W. Fellner. Src-a streamable format for generalized web-based 3d data transmission. In *Proceedings of the Nineteenth International ACM Conference on 3D Web Technologies*, pages 35–43. ACM, 2014.

[LTZH09] Longlong Liao, Guoxin Tan, Zheng Zhong, and Tinglei Hao. The design and implementation of a management platform for creating 3d digital content production. In *Education Technology and Computer Science, 2009. ETCS'09. First International Workshop on*, volume 2, pages 637–642. IEEE, 2009.

[LWS+13] Max Limper, Stefan Wagner, Christian Stein, Yvonne Jung, and André Stork. Fast delivery of 3d web content: A case study. In *Proceedings of the 18th International Conference on 3D Web Technology*, Web3D '13, pages 11–17, New York, NY, USA, 2013. ACM.

[MAB+09] Grzegorz Malewicz, Matthew H. Austern, Aart J.C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 28th ACM symposium on Principles of distributed computing*, pages 6–6, New York, NY, USA, 2009. ACM.

[Mac06] Matt Mackall. Towards a better scm: Revlog and mercurial. In *Linux Symposium*, volume 2, page 83, Ottawa, Ontario, Canada, July 2006.

[Mal92] D. H. Maling. *Coordinate Systems and Map Projections*. Pergamon, second edition, January 1992. ISBN-10: 0080372333.

[Man12] Carl Manneh. Mojang and UN presents: Block by block, September 2012. URL: https://mojang.com/2012/09/mojang-and-un-presents-block-by-block.

[Mar02] Joe Marini. *Document Object Model*. McGraw-Hill, Inc., New York, NY, USA, first edition, 2002.

[Mar07] Howie Markson. Achieving cad interoperability in global product design environments. *White Paper, SpaceClaim Corporation*, 2007.

[Mar10] Drake Martinet. Dive tech-onlive now more than just a game. *All Things Digital — Voices*, December 2010.

[Mar11] Chris Marrin. Webgl specification v1.0. Technical specification, Khronos Group, February 2011.

[Mar13] Jan Marsch. Osm buildings, 2013. Computer Software, URL: http://osmbuildings.org.

[McC11] Brian McClendon. Step inside the map with google mapsgl, October 2011. URL: http://googleblog.blogspot.co.uk/2011/10/step-inside-map-with-google-mapsgl.html.

[McE11] Cameron McEfee. Behold: Image view modes, March 2011. URL: https://github.com/blog/817-behold-image-view-modes.

[Mej09] M. J. Mejose. Art diff, December 2009. Computer Software, URL: http://code.google.com/p/artdiff.

[Men02] Tom Mens. A state-of-the-art survey on software merging. *IEEE Transactions on Software Engineering*, 28(5):449–462, 2002.

[Met06] MetaCarta. Openlayers, January 2006. Computer Software, URL: http://www.openlayers.org.

[MF08] Chip Morningstar and F Randall Farmer. The lessons of lucasfilm's habitat. *Journal For Virtual Worlds Research*, 1(1), 2008.

[MGT+03] Tamara Munzner, François Guimbretière, Serdar Tasiran, Li Zhang, and Yunhong Zhou. Treejuxtaposer: scalable tree comparison using focus+context with guaranteed visibility. *ACM Trans. Graph.*, 22(3):453–462, July 2003.

[MH10] Michael Mclaughlin and John M. Harper. *Oracle Database 11g PL/SQL Programming Workbook (Oracle Press)*. McGraw-Hill Osborne, workbook edition, March 2010. ISBN-10: 0071493697.

[MHUC12] Marc Manzano, José Alberto Hernández, M Uruenña, and Eusebi Calle. An empirical study of cloud gaming. In *Proceedings of the 11th Annual Workshop on Network and Systems Support for Games*, page 17. IEEE Press, 2012.

[Mic10] Microsoft Corporation. Bing maps, 2010. Computer Software, URL: http://www.bing.com/maps.

[Mic11] Microsoft Corporation. Microsoft remotefx, February 2011. URL: http://technet.microsoft.com/en-us/library/ff817578(v=ws.10).aspx.

[Mic14] Microsoft Developer Network. [ms-rdpbcgr]: Remote desktop protocol: Basic connectivity and graphics remoting, May 2014. URL: http://msdn.microsoft.com/en-us/library/cc240445.aspx.

[Mig14] Migenius Pty LTD. Interactive photorealism for the web, January 2014. URL: http://www.migenius.com/products/realityserver/overview.

[MKFC01] Takashi Michikawa, Takashi Kanai, Masahiro Fujita, and Hiroaki Chiyokura. Multiresolution interpolation meshes. In *Computer Graphics and Applications, 2001. Proceedings. Ninth Pacific Conference on*, pages 60–69. IEEE, 2001.

[ML10] Aaftab Munshi and Jon Leech. Opengl es common profile specification. Technical specification, Khronos Group, November 2010.

[MMP09] Miriah Meyer, Tamara Munzner, and Hanspeter Pfister. Mizbee: a multiscale synteny browser. *Visualization and Computer Graphics, IEEE Transactions on*, 15(6):897–904, 2009.

[Mon14a] MongoDB, Inc. BSON–binary json specification. Technical specification, MongoDB, Inc., January 2014.

[Mon14b] MongoDB, Inc. Javascript + C++ BSON parser, January 2014. Computer Software, URL: https://github.com/mongodb/js-bson.

[MPH10] Peter Membrey, Eelco Plugge, and Tim Hawkins. *The Definitive Guide to MongoDB: The NoSQL Database for Cloud & Desktop Computing*. APRESS ACADEMIC, first edition, 2010.

[MPSR01] David McWherter, Mitchell Peabody, Ali C Shokoufandeh, and William Regli. Database techniques for archival of solid models. In *Proceedings of the sixth ACM symposium on Solid modeling and applications*, pages 78–87. ACM, 2001.

[MPWC13] Niloy J. Mitra, Mark Pauly, Michael Wand, and Duygu Ceylan. Symmetry in 3D Geometry: Extraction and Applications. *Computer Graphics Forum*, 32(6):1–23, 2013.

[MSG11] Christophe Mouton, Kristian Sons, and Ian Grimstead. Collaborative visualization: current systems and future trends. In *Proceedings of the 16th International Conference on 3D Web Technology*, pages 101–110. ACM, 2011.

[MT95] Duncan C. Miller and Jack A. Thorpe. Simnet: The advent of simulator networking. *Proceedings of the IEEE*, 83(8):1114–1123, 1995.

[MXLH13] Min Meng, Jiazhi Xia, Jun Luo, and Ying He. Unsupervised co-segmentation for 3d shapes using iterative multi-label optimization. *Computer-Aided Design*, 45(2):312–320, 2013.

[NBCW+11] Andy Nguyen, Mirela Ben-Chen, Katarzyna Welnicka, Yinyu Ye, and Leonidas Guibas. An optimization approach to improving collections of shape maps. In *Computer Graphics Forum*, volume 30, pages 1481–1491. Wiley Online Library, 2011.

[Nei94] Jakob Neilsen. *Usability Engineering (Interactive Technologies)*. Morgan Kaufmann, November 1994. ISBN-10: 0125184069.

[New07] Richard G. Newell. The why and the how of the long transaction, July 2007.

[NH82] Thomas Neumann and Christoph Hornung. Consistency and transactions in cad database. In *Proceedings of the 8th International Conference on Very Large Data Bases*, VLDB '82, pages 181–188, San Francisco, CA, USA, 1982. Morgan Kaufmann Publishers Inc.

[NH06] Bonnie Nardi and Justin Harris. Strangers and friends: Collaborative play in world of warcraft. In *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, pages 149–158. ACM, 2006.

[Nin08] Tateru Nino. Second life grid closed due to asset problems, January 2008.

[Nok13] Nokia Corporation. Here, 2013. Computer Software, URL: http://here.com.

[NVI09] NVIDIA ARC GmbH. Nvidia advanced rendering: Nvidia iray - mental ray, October 2009. URL: http://www.nvidia-arc.com/iray.html.

[Obj11] Object Management Group. Mof 2 xmi mapping (xmi) v2.4.1, Aug 2011. URL: http://www.omg.org/spec/XMI.

[OH11] Regina O. Obe and Leo S. Hsu. *PostGIS in Action*. Manning Publications, first edition, April 2011. ISBN-10: 1935182269.

[OHL+08] John D. Owens, Mike Houston, David Luebke, Simon Green, John E. Stone, and James C. Phillips. Gpu computing. *Proceedings of the IEEE*, 96(5):879–899, 2008.

[Olb12] Manuel Olbrich. Accessing http interfaces within x3d script nodes. In *Proceedings of the 17th International Conference on 3D Web Technology*, Web3D '12, pages 139–142, NY, USA, 2012. ACM.

[OnL09] OnLive, Inc. Onlive revolutionizes video games: State-of-the-art games play over the internet without high-end hardware. Press release, OnLive, San Francisco, CA, March 2009.

[Ope03] Opera Software ASA. Presto, January 2003. Software Library.

[Ope11] Open Geospatial Consortium. Open geospatial consortium standards, September 2011. URL: http://www.opengeospatial.org/standards.

[O'R85] Joseph O'Rourke. Finding minimal enclosing boxes. *International Journal of Computer & Information Sciences*, 14(3):183–199, 1985.

[Ora12a] Oracle Corporation. Liveconnect support in the new Java™ plug-in technology. White paper, Java SE 6 update 10, March 2012. URL: http://jdk6.java.net/plugin2/liveconnect.

[Ora12b] Oracle Corporation. Oracle spatial & oracle locator, location features for oracle database 11g, January 2012. Computer Software, URL: http://www.oracle.com/technetwork/database/options/spatialandgraph.

[Ora12c] Oracle Corporation. Oracle workspace manager, January 2012. Computer Software, URL: http://www.oracle.com/technetwork/database/enterprise-edition/index-087067.html.

[OSNZ10] M. Over, A. Schilling, S. Neubauer, and A. Zipf. Generating web-based 3d city models from openstreetmap: The current situation in germany. *Computers, Environment and Urban Systems*, 34(6):496–507, 2010.

[Ove11] Ove Arup & Partners. Collaborative map, January 2011. Computer Software, URL: http://collaborativemap.org.

[Ove12] Ove Arup & Partners. King's cross station, March 2012. URL: http://www.arup.com/Projects/Kings_Cross_Station.

[PA11] Tony Parisi and Rémi Arnaud. 3D REST 3D specification v0.2, April 2011. URL: http://rest3d.org.

[Pal13] Todd Palamar. *Mastering Autodesk Maya 2014: Autodesk Official Press*. Sybex, first edition, July 2013. ISBN-10: 1118574966.

[Par12] Ben Parfitt. 1,600 users - the story of onlive's collapse starts to emerge, August 2012. URL: http://www.mcvuk.com/news/read/1-600-users-the-story-of-onlive-s-collapse-starts-to-emerge/0101506.

[Pat11] Nirav Patel. Thingidiff: Visualizing 3d model diffs with thingiview.js, October 2011. Available online: http://eclecti.cc/computergraphics/thingidiff-visualizing-3d-model-diffs-with-thingiview-js.

[Pau05] Linda Dailey Paulson. Building rich web applications with ajax. *Computer*, 38(10):14–17, 2005.

[PCSF08] Michael Pilato, Ben Collins-Sussman, and Brian W. Fitzpatrick. *Version Control with Subversion*. O'Reilly Media, second edition, September 2008. ISBN-10: 0596510330.

[PG07] Renato Pajarola and Enrico Gobbetti. Survey of semi-regular multiresolution models for interactive terrain rendering. *The Visual Computer*, 23(8):583–605, 2007.

[Pil10] Mark Pilgrim. *HTML5: Up and Running*. O'Reilly Media, Inc., 1st edition, 2010.

[PMW+08] Mark Pauly, Niloy J. Mitra, Johannes Wallner, Helmut Pottmann, and Leonidas J. Guibas. Discovering structural regularity in 3d geometry. *ACM Trans. Graph.*, 27(3):43:1–43:11, August 2008.

[Pos80] Jon Postel. User datagram protocol. *Isi*, 1980.

[Pos81] Jon Postel. Transmission control protocol. *The Internet Engineering Task Force*, September 1981.

[PSZ06] Christine Parent, Stefano Spaccapietra, and Esteban Zimányi. *Conceptual Modeling for Traditional and Spatio-Temporal Applications*. Springer, 2006.

[PT97] Les Piegl and Wayne Tiller. Curve and surface basics. In *The NURBS Book*, Monographs in Visual Communication, pages 1–46. Springer Berlin Heidelberg, 1997.

[PTMH05] Dimitris Papadias, Yufei Tao, Kyriakos Mouratidis, and Chun Kit Hui. Aggregate nearest neighbor queries in spatial databases. *ACM Transactions on Database Systems (TODS)*, 30(2):529–576, 2005.

[Rag94] David Raggett. Extending www to support platform independent virtual reality. In *Proceedings of the 5th Joint European Networking Conference (JENC5)*, volume 464, page 2, 1994.

[Rau12] Guillermo Rauch. *Smashing Node.js: JavaScript Everywhere*. Wiley, second edition, September 2012. ISBN-10: 1119962595.

[Rei02] Dirk Reiners. *OpenSG: A scene graph system for flexible and efficient realtime rendering for virtual and augmented reality applications*. Dissertation, Fraunhofer IGD, 2002.

[Res03] Cyon Research. The building information model: A look at graphisofts virtual building concept. White paper, Cyon Research Corporation, January 2003.

[RFH+14] Edward Red, David French, Ammon Hepworth, Greg Jensen, and Brett Stone. Multiuser computer-aided design and engineering software applications. In Dirk Schaefer, editor, *Cloud-Based Design and Manufacturing (CBDM)*, pages 25–62. Springer International Publishing, 2014.

[RGAM10] Nick Randolph, David Gardner, Chris Anderson, and Michael Minutillo. *Professional Visual Studio 2010*. John Wiley & Sons, 2010.

[RGG03] Raghu Ramakrishnan, Johannes Gehrke, and Johannes Gehrke. *Database management systems*, volume 3. McGraw-Hill New York, 2003.

[RH94] John Rohlf and James Helman. Iris performer: a high performance multiprocessing toolkit for real-time 3d graphics. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 381–394. ACM, 1994.

[Rib14] Laurent Ribon. Opengl library class (glc lib), January 2014. Computer Software, URL: https://github.com/laumaya/GLC_lib.

[Ris13] Ray Rischpater. *Application Development with Qt Creator*. Packt Publishing, first edition, November 2013. ISBN-10: 1783282312.

[Ros88] L. E. Roscoe. Stereolithography interface specification. *3D Systems, Inc.*, 1988.

[RPO12] Fabrice Robinet, Tony Parisi, and Patrick Ozzi. glTF, 2012. Computer Software, URL: https://github.com/KhronosGroup/collada2json/wiki/glTF.

[RR07] Leonard Richardson and Sam Ruby. *Restful web services*. O'Reilly Media, Inc., first edition, 2007.

[RSV01] Philippe Rigaux, Michel Scholl, and Agnès Voisard. *Spatial Databases: With Application to GIS*. Morgan Kaufmann, 2001. ISBN: 978-1558605886.

[RVD+11] Ton Roosendaal, Dolf Veenliet, Jeremy Davidson, William Reynish, Beorn Leonard, and Pablo Vazquez. Sintel svn repository, July 2011. URL: http://download.blender.org/durian/svn.

[RW98] Tristan Richardson and Kenneth R. Wood. The rfb protocol, January 1998.

[RW12] Eric Redmond and Jim R. Wilson. *Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement*. Pragmatic Bookshelf, first edition, May 2012. ISBN-10: 1934356921.

[RWE13] Ian Robinson, Jim Webber, and Emil Eifrem. *Graph Databases*. O'Reilly Media, first edition, June 2013. ISBN-10: 1449356265.

[Rym07] Michael Rymaszewski. *Second life: The official guide*. John Wiley & Sons, 2007.

[SA86] Scott M. Staley and David C. Anderson. Functional specification for cad databases. *Computer-aided design*, 18(3):132–138, 1986.

[SA94] Mark Segal and Kurt Akeley. The design of the opengl graphics interface. In *Silicon Graphics Computer Systems*. Citeseer, 1994.

[Sat93] Mahadev Satyanarayanan. Distributed file systems. *Distributed Systems. Addison-Wesley and ACM Press,*, 821:145–154, 1993.

[SBM86] David Shuey, David Bailey, and Thomas P Morrissey. Phigs: A standard, dynamic, interactive graphics interface. *Computer Graphics and Applications, IEEE*, 6(8):50–57, 1986.

[SBSCO06] Andrei Sharf, Marina Blumenkrants, Ariel Shamir, and Daniel Cohen-Or. Snappaste: an interactive technique for easy mesh composition. *Vis. Comput.*, 22(9):835–844, September 2006.

[SBU+10] Andreas Schiefer, René Berndt, Torsten Ullrich, Volker Settgast, and Dieter W. Fellner. Service-oriented scene graph manipulation. In *Proceedings of the 15th International Conference on Web 3D Technology*, Web3D '10, pages 55–62, NYC, NY, USA, 2010. ACM.

[SC92] Paul S. Strauss and Rikk Carey. An object-oriented 3d graphics toolkit. *SIGGRAPH Comput. Graph.*, 26(2):341–349, July 1992.

[SC03] Shashi Shekhar and Sanjay Chawla. *Spatial databases: a tour*, volume 2003. prentice hall Upper Saddle River, NJ, 2003.

[SCFY96] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *Computer*, 29(2):38–47, 1996.

[Sch97] Ralph Schroeder. Networked worlds: social aspects of multi-user virtual reality technology. *Sociological Research Online*, 2(4), 1997.

[Sch10] Ryan Michael Schmidt. *Part-Based Representation and Editing of 3D Surface Models*. PhD thesis, University of Toronto, Canada, 2010.

[Sch13] Alexander Schreyer. *Architectural Design with SketchUp: Component-based Modeling, Plugins, Rendering and Scripting*. John Wiley & Sons, first edition, January 2013. ISBN-10: 1118123093.

[Sch14] Herbert Schildt. *Java: The Complete Reference*. McGraw-Hill Osborne Media, ninth edition, March 2014. ISBN-10: 0071808558.

[SCR+99] Shashi Shekhar, Sanjay Chawla, Sivakumar Ravada, Andrew Fetterer, Xuan Liu, and Chang-tien Lu. Spatial databases-accomplishments and research needs. *Knowledge and Data Engineering, IEEE Transactions on*, 11(1):45–55, 1999.

[SFF00] Sheryl Staub-French and Martin Fischer. Practical and research issues in using industry foundation classes for construction cost estimating, 2000.

[SG86] Robert W. Scheifler and Jim Gettys. The x window system. *ACM Transactions on Graphics (TOG)*, 5(2):79–109, 1986.

[SGK+14] Thomas Schulze, Alexander Gessler, Kim Kulling, David Nadlinger, Jonathan Klein, Mark Sibly, and Matthias Gubisch. Open asset import library (assimp), January 2014. Computer Software, URL: https://github.com/assimp/assimp.

[SGLS93] Chris Shaw, Mark Green, Jiandong Liang, and Yunqi Sun. Decoupled simulation in virtual reality with the mr toolkit. *ACM Transactions on Information Systems (TOIS)*, 11(3):287–317, 1993.

[SH12] René Schubotz and Andreas Harth. Towards networked linked data-driven web3d applications. In *Dec3D*, 2012.

[SHBL06] Nigel Shadbolt, Wendy Hall, and Tim Berners-Lee. The semantic web revisited. *Intelligent Systems, IEEE*, 21(3):96–101, 2006.

[Sho12] Daniel Short. Teaching scientific concepts using a virtual world–minecraft. *Teaching Science*, 58(3):55–58, 2012.

[Shr10] Ryan Shrout. Onlive game service preview - is this the future of pc gaming? *PC Perspective*, January 2010.

[Sil12] Jonas Raoni Soares Silva. Js-bson, March 2012. Computer Software.

[Sim10] Alvise Simondetti. Built environment modelling handbook. Arup Group Limited, June 2010.

[Sim12] Alvise Simondetti. Digital environments for experiential design - enhancing designers perception. Arup Group Limited, September 2012.

[Sin11] Eric Sink. *Version Control by Example*. Pyrenean Gold Press, first edition, 2011. ISBN-10: 0983507902.

[Ska13] Mike Skalnik. 3d file diffs, September 2013. URL: https://github.com/blog/1633-3d-file-diffs.

[SKR+10] Kristian Sons, Felix Klein, Dmitri Rubinstein, Sergiy Byelozyorov, and Philipp Slusallek. XML3D: Interactive 3D Graphics for the Web. In *Proceedings of the*

15th International Conference on Web 3D Technology, Web3D '10, pages 175–184, New York, NY, USA, 2010. ACM.

[SMKF04] Philip Shilane, Patrick Min, Michael Kazhdan, and Thomas Funkhouser. The Princeton shape benchmark. In *Shape Modeling International*, June 2004.

[SML06] Will Schroeder, Ken Martin, and Bill Lorensen. *Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*. Kitware, fourth edition, December 2006. ISBN-10: 193093419X.

[SNF10] Christian Stab, Kawa Nazemi, and Dieter W. Fellner. Sematime - timeline visualization of time-dependent relations and semantics. In *Proc ISVC*, pages 514–523, 2010.

[SO09] Anthony Steed and Manuel Oliveira. *Networked Graphics: Building Networked Games and Virtual Environments*. Elsevier, 2009. ISBN-10: 0123744237.

[Spe11] Scott Spencer. *ZBrush Character Creation: Advanced Digital Sculpting*. John Wiley & Sons, second edition, February 2011. ISBN-10: 0470572574.

[SS10] Ryan Schmidt and Karan Singh. meshmixer: an interface for rapid mesh composition. In *ACM SIGGRAPH 2010 Talks*, SIGGRAPH '10, pages 6:1–6:1, New York, NY, USA, 2010. ACM.

[SS11] Herbert Stocker and Peter Schickel. X3D binary encoding results for free viewpoint networked distribution and synchronization. In *Proceedings of the 16th International Conference on 3D Web Technology*, Web3D '11, pages 67–70, New York, NY, USA, 2011. ACM.

[SS12] William Steptoe and Anthony Steed. Multimodal data capture and analysis of interaction in immersive collaborative virtual environments. *Presence: Teleoperators and Virtual Environments*, 21(4):388–405, 2012.

[SSCO08] Lior Shapira, Ariel Shamir, and Daniel Cohen-Or. Consistent mesh partitioning and skeletonisation using the shape diameter function. *Vis. Comput.*, 24(4):249–259, 2008.

[SSK+13] Kristian Sons, Christian Schlinkmann, Felix Klein, Dmitri Rubinstein, and Philipp Slusallek. xml3d.js: Architecture of a Polyfill Implementation of XML3D. In *Software Engineering and Architectures for Realtime Interactive Systems (SEARIS), 2013 6th Workshop on (to appear)*, 2013.

[SSO+11] Anthony Steed, William Steptoe, Wole Oyekoya, Fabrizio Pece, Tim Weyrich, Jan Kautz, Doron Friedman, Angelika Peer, Massimiliano Solazzi, Franco Tecchia, et al. Beaming: an asymmetric telepresence system. *IEEE computer graphics and applications*, 32(6):10–17, 2011.

[SSS+10] Lior Shapira, Shy Shalom, Ariel Shamir, Daniel Cohen-Or, and Hao Zhang. Contextual part analogies in 3d objects. *Int. J. Comput. Vision*, 89(2-3):309–326, September 2010.

[SSS14] Jan Sutter, Kristian Sons, and Philipp Slusallek. Blast: a binary large structured transmission format for the web. In *Proceedings of the Nineteenth International ACM Conference on 3D Web Technologies*, pages 45–52. ACM, 2014.

[Ste92] Neal Stephenson. *Snow Crash*. Bantam Books, first edition, 1992. ISBN 0-553-08853-X.

[Ste10] Tim Stevens. Onlive microconsole torn down, marvell armada found lurking within. *engadget*, 2010. URL: http://www.engadget.com/2010/12/14/onlive-microconsole-torn-down-marvell-armada-found-lurking-with.

[SV12] David Sowder and David Vierra. Mcedit, October 2012. Computer Software, URL: http://www.mcedit.net.

[SvKK+11] Oana Sidi, Oliver van Kaick, Yanir Kleiman, Hao Zhang, and Daniel Cohen-Or. Unsupervised co-segmentation of a set of shapes via descriptor-space spectral clustering. *ACM SIGGRAPH Asia*, 30(6):126:1–126:9, 2011.

[SZGP05] Robert W. Sumner, Matthias Zwicker, Craig Gotsman, and Jovan Popović. Mesh-based inverse kinematics. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 488–495. ACM, 2005.

[SZT12] Baron Schwartz, Peter Zaitsev, and Vadim Tkachenko. *High Performance MySQL: Optimization, Backups, and Replication*. O'Reilly Media, third edition, April 2012. ISBN-10: 1449314287.

[Tam10] Tony Tam. 12 months with mongodb, October 2010. URL: http://blog.wordnik.com/12-months-with-mongodb.

[Tan07] Andrew S. Tanenbaum. *Modern operating systems*. Prentice Hall Press, 2007.

[Tat13] Kevin Tatroe. *Programming PHP*. O'Reilly Media, third edition, February 2013. ISBN-10: 1449392776.

[TBT+87] Jack A. Thorpe, Gary W. Bloedorn, Richard Taylor, Duncan C. Miller, and Michael Cyrus. The simnet network and protocol. Technical report, DTIC Document, 1987.

[Tel05] Telecommunication Standardization Sector. Information technology - Generic applications of ASN.1: Fast infoset. Rec, ITU, May 2005. X.891, ISO/IEC 24824-1:2007 (Common).

[Tel08] Telecommunication Standardization Sector. Information technology - open systems interconnection - procedures for the operation of osi registration authorities: Generation and registration of universally unique identifiers (uuids) and their use as asn.1 object identifier components. Recommendation, Internation Telecommunications Union, August 2008. Rec. ITU-T X.667, ISO/IEC 9834-8.

[Tes13] Claudio Tesoriero. *Getting Started with OrientDB*. PACKT PUBLISHING, August 2013. ISBN-10: 1782169954.

[The13] The British Standards Institution. Pas 1192-2:2013 specification for information management for the capital/delivery phase of construction projects using building information modelling. Technical specification, BSI Standards Limited, February 2013.

[Tho11] Craig W. Thompson. Next-generation virtual worlds: architecture, status, and directions. *Internet Computing, IEEE*, 15(1):60–65, 2011.

[Tor14] Linus Torvalds. Linux kernel 3.x git repository, May 2014. [Online; accessed 13-May-2014], URL: https://github.com/torvalds/linux.

[TPZB08] Andrew Tan, Binh Pham, Jinglan Zhang, and Ross Brown. A collaborative framework for simultaneous and seamless 3d graphics manipulation. In *Proceedings of the 6th International Conference on Advances in Mobile Computing and Multimedia*, pages 206–210. ACM, 2008.

[Tri14] Trimble Navigation Ltd. Sketchup 3d warehouse, June 2014. URL: https://3dwarehouse.sketchup.com.

[Tur14] TurboSquid Ltd. Turbosquid, June 2014. URL: http://www.turbosquid.com.

[TV08] Johan W. H. Tangelder and Remco C. Veltkamp. A survey of content based 3d shape retrieval methods. *Multimedia tools and applications*, 39(3):441–471, 2008.

[TW87] Andrew S. Tanenbaum and Albert Woodhull. *Operating systems: design and implementation*, volume 2. Prentice-Hall Englewood Cliffs, NJ, 1987.

[TZ09] Siak Chuan Tan and Jinglan Zhang. Dynamic lock synchronisation for collaborative 3d applications. In *Proceedings of the 7th International Conference on Advances in Mobile Computing and Multimedia*, MoMM '09, pages 540–544, New York, NY, USA, 2009. ACM.

[Uni10] Unity Technologies. Using external version control systems with unity, October 2010. Computer Software.

[UNI11] UNIT4 N.V. Unit4 business collaborator, April 2011. Computer Software.

[Ves03] Jennifer Vesperman. *Essential CVS*. O'Reilly Media, first edition, June 2003. ISBN-10: 0596004591.

[Vie09] John Viega. Cloud computing and the common man. *Computer*, 42(8):106–108, 2009.

[Vis12] VisTrails, Inc. Vistrails provenance explorer for maya, 2012. Computer Software, URL: http://www.vistrails.com/maya.html.

[VKR13] James Vandezande, Eddy Krygiel, and Phil Read. *Mastering Autodesk Revit Architecture 2014: Autodesk Official Press*. John Wiley & Sons, first edition, June 2013. ISBN-10: 1118521307.

[VKZHCO11] Oliver Van Kaick, Hao Zhang, Ghassan Hamarneh, and Daniel Cohen-Or. A survey on shape correspondence. *(CGF)*, 30(6):1681–1707, 2011.

[VMZ+10] Chad Vicknair, Michael Macias, Zhendong Zhao, Xiaofei Nan, Yixin Chen, and Dawn Wilkins. A comparison of a graph database and a relational database: a data provenance perspective. In *Proceedings of the 48th annual Southeast regional conference*, page 42. ACM, 2010.

[VP03] Athena Vakali and George Pallis. Content delivery networks: Status and trends. *Internet Computing, IEEE*, 7(6):68–74, 2003.

[WAvK+12] Yunhai Wang, Shmulik Asafi, Oliver van Kaick, Hao Zhang, Daniel Cohen-Or, and Baoquan Chen. Active co-analysis of a set of shapes. *ACM Transactions on Graphics (TOG)*, 31(6):165, 2012.

[WBWK00] Michelle Q. Wang Baldonado, Allison Woodruff, and Allan Kuchinsky. Guidelines for using multiple views in information visualization. In *Proceedings of the working conference on Advanced visual interfaces*, AVI '00, pages 110–119, New York, NY, USA, 2000. ACM.

[Wen14] Richard Wentk. *Xcode 5 Developer Reference*. John Wiley & Sons, first edition, June 2014. ISBN-10: 111883433X.

[Wer94] Josie Wernecke. *The Inventor Toolmaker: Extending Open Inventor, Release 2*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 1994.

[Win05] Laura Wingerd. *Practical Perforce*. O'Reilly Media, first edition, 2005. ISBN-10: 0596101856.

[WK96] Jürgen Wäsch and Wolfgang Klas. History merging as a mechanism for concurrency control in cooperative environments. In *Proceedings of the 6th International Workshop on Research Issues in Data Engineering (RIDE '96) Interoperability of Nontra-*

ditional Database Systems, pages 76–, Washington, DC, USA, 1996. IEEE Computer Society.

[Wor12] World Wide Web Consortium (W3C). Http archive (har) format. Editor's draft, W3C, August 2012.

[WWS+13] Zizhao Wu, Yunhai Wang, Ruyang Shou, Baoquan Chen, and Xinguo Liu. Unsupervised co-segmentation of 3d shapes via affinity aggregation spectral clustering. *Computers & Graphics*, 37(6):628–637, 2013.

[Yag12] Takaaki Yagiu. *Modeling Design Objects and Processes*. Springer Publishing Company, Incorporated, 1st edition, 2012.

[Yer96] Franois Yergeau. Utf-8, a transformation format of unicode and iso 10646. *The Internet Engineering Task Force*, October 1996.

[Zha09] Andy Zhang. *Beginning Mark Logic with XQuery and MarkLogic Server*. Champion Writers, June 2009. ISBN-10: 1608300153.

[ZHC+00] Bob Zeleznik, Loring Holden, Michael Capps, Howard Abrams, and Tim Miller. Scene-graph-as-bus: Collaboration between heterogeneous stand-alone 3-d graphical applications. In *Eurographics*, 2000.

[ZHV12] Zhou Zhao, Kai Hwang, and Jose Villeta. Game cloud design with virtualized cpu/gpu servers and initial performance results. In *Proceedings of the 3rd Workshop on Scientific Cloud Computing Date*, ScienceCloud '12, pages 23–30, New York, NY, USA, 2012. ACM.

[ZKS11] Loutfouz Zaman, Ashish Kalra, and Wolfgang Stuerzlinger. The effect of animation, dual view, difference layers, and relative re-layout in hierarchical diagram differencing. In *GI '11*, pages 183–190, 2011.

[ZP05] Sisi Zlatanova and David Prosperi. *Large-scale 3D data integration: challenges and opportunities*. CRC Press, 2005.

[ZS09] Aditya Zutshi and Geetika Sharma. A study of virtual environments for enterprise collaboration. In *Proceedings of the 8th International Conference on Virtual Reality Continuum and its Applications in Industry*, VRCAI '09, pages 331–333, New York, NY, USA, 2009. ACM.

[ZYLH14] Jianping Zhang, Fangqiang Yu, Ding Li, and Zhenzhong Hu. Development and implementation of an industry foundation classes-based graphic information model for virtual construction. *Computer-Aided Civil and Infrastructure Engineering*, 29(1):60–74, 2014.