PolyVR - A Virtual Reality Authoring Framework for Engineering Applications

Zur Erlangung des akademischen Grades
Doktor der Ingenieurwissenschaften
von der KIT-Fakultät für Maschinenbau
des Karlsruher Instituts für Technologie (KIT)

genehmigte **Dissertation**
von

Dipl.-Phys. Victor Häfner

Mündlichen Prüfung:	10.09.2019
Referent:	Prof. Dr. Dr.-Ing. Dr. h. c. Jivka Ovtcharova
Korreferent:	Prof. Dr.-Ing Fahmi Bellalouna

Abstract

Virtual reality is a marvelous place, free of constraints and plenty of opportunities. It is the perfect space for experiencing science and engineering, but there is a lack of infrastructure to make virtual reality more accessible, especially for engineering applications. This work aims at creating a software environment that enables an easier development of virtual reality applications and their deployment on immersive hardware setups. Virtual engineering, the use of virtual environments for design reviews during the product development process, is still heavily underutilized, especially in small and medium sized enterprises. The main reasons are still the costs related to using such technology, but the costs have shifted from virtual reality hardware to high maintenance and software development costs. Especially the lack of automated virtualization work-flows hinders the adoption of virtual reality solutions. One important aspect of automating virtualization is infusing intelligence in artificial environments. This can be achieved using knowledge bases and ontologies. Ontologies are the basis of human understanding and intelligence. Categorization of our universe in concepts, properties and rules is fundamental to processes like observation, learning or knowing.

This work aims at developing a stepping stone towards a broader deployment of virtual reality applications in all areas of science and engineering. The approach taken is to build up a virtual reality authoring tool, a software framework aimed at easing the creation of virtual worlds, and the deployment of those worlds in advanced immersive hardware environments like distributed visualization systems. A further goal of this thesis is to enable the intuitive authoring of semantic elements in virtual worlds. This has the potential to revolutionize the process of creating virtual content and the interaction possibilities. Smart immersive environments are the key to foster learning and training in virtual worlds, planning and monitoring of processes or pave the way for completely new interaction paradigms.

Kurzfassung

Die virtuelle Realität ist ein fantastischer Ort, frei von Einschränkungen und vielen Möglichkeiten. Für Ingenieure ist dies der perfekte Ort, um Wissenschaft und Technik zu erleben, es fehlt jedoch die Infrastruktur, um die virtuelle Realität zugänglich zu machen, insbesondere für technische Anwendungen. Diese Arbeit bescheibt die Entstehung einer Softwareumgebung, die eine einfachere Entwicklung von Virtual-Reality-Anwendungen und deren Implementierung in immersiven Hardware-Setups ermöglicht. Virtual Engineering, die Verwendung virtueller Umgebungen für Design-Reviews während des Produktentwicklungsprozesses, wird insbesondere von kleinen und mittleren Unternehmen nur äußerst selten eingesetzt. Die Hauptgründe sind nicht mehr die hohen Kosten für professionelle Virtual-Reality-Hardware, sondern das Fehlen automatisierter Virtualisierungsabläufe und die hohen Wartungs- und Softwareentwicklungskosten. Ein wichtiger Aspekt bei der Automatisierung von Virtualisierung ist die Integration von Intelligenz in künstlichen Umgebungen. Ontologien sind die Grundlage des menschlichen Verstehens und der Intelligenz. Die Kategorisierung unseres Universums in Begriffe, Eigenschaften und Regeln ist ein grundlegender Schritt von Prozessen wie Beobachtung, Lernen oder Wissen.

Diese Arbeit zielt darauf ab, einen Schritt zu einem breiteren Einsatz von Virtual-Reality-Anwendungen in allen Bereichen der Wissenschaft und Technik zu entwickeln. Der Ansatz ist der Aufbau eines Virtual-Reality-Authoring-Tools, eines Softwarepakets zur Vereinfachung der Erstellung von virtuellen Welten und der Implementierung dieser Welten in fortschrittlichen immersiven Hardware-Umgebungen wie verteilten Visualisierungssystemen. Ein weiteres Ziel dieser Arbeit ist es, das intuitive Authoring von semantischen Elementen in virtuellen Welten zu ermöglichen. Dies sollte die Erstellung von virtuellen Inhalten und die Interaktionsmöglichkeiten revolutionieren. Intelligente immersive Umgebungen sind der Schlüssel, um das Lernen und Trainieren in virtuellen Welten zu fördern, Prozesse zu planen und zu überwachen oder den Weg für völlig neue Interaktionsparadigmen zu ebnen.

Acknowledgements

I would like to express my deep gratitude to Prof. Dr. Dr.-Ing. Dr. h. c. Jivka Ovtcharova for the opportunities and support during my time as research assistant at the Institute for Information Management in engineering at the Karlsruhe Institute of Technology. I would like to thank Prof. Dr.-Ing Fahmi Bellalouna for his cooperation and support as well as his willingness to co-supervise this work. Special thanks are due to my colleagues with whom I worked closely together on so many projects. I wish to thank also the numerous students who gave me their trust and invested their time, for discovering and evolving together, for exploring the many applications of virtual reality and helped me shape through many iterations the foundations for this work.

During my time as a student and researcher, I met my beloved wife Polina, with whom I walked together this path of exploration, research and innovation. Our daughter Leonie is born while finalizing this thesis and thus I dedicate this work to both of them. Finally, I wish to thank my parents and my sister for their support and encouragement throughout my studies.

Karlsruhe, September 2019 *Victor Häfner*

Inhaltsverzeichnis

Abkürzungs- und Symbolverzeichnis

AES	Advanced Encryption Standard
AG	AktienGesellschaft (limited company)
AI	Artificial Intelligence
AP	STEP Application Protocol
API	Application Programming Interface
ART	Advanced Realtime Tracking
ASCII	American Standard Code for Information Interchange
BCAD	Building CAD
BIM	Building Information Modeling
BREP	Boundary REPresentation
CAD	Computer-Aided Design
CAE	Computer-Aided Engineering
CAM	Computer-Aided Manufacturing
CAN	Controller Area Network
CAVE	Cave automatic virtual environment
CEF	Chromium Embedded Framework
CG	C for Graphics, high-level shading language
CGAL	The Computational Geometry Algorithms Library
CPU	Central Processing Unit
CSG	Constructive Solid Geometry
CSS	Cascading Style Sheets
DLL	Dynamic-Link Library
DLR	Deutsches Zentrum für Luft- und Raumfahrt (German Aerospace Center)
DoF	Degrees of Freedom
DXF	Drawing eXchange Format
ECAD	Electrical CAD
ECU	Engine Control Unit
ERP	Enterprise Resource Planning
FBX	FilmBoX (Autodesk file format)
FEM	Finite Element Method
FM	Frequency Modulation synthesis

GDAL	Geospatial Data Abstraction Library
GIL	Global Interpreter Lock
GIS	Geographic Information System
GLSL	OpenGL Shading Language
GMBH	Gesellschaft Mit Beschränkter Haftung (private limited company)
GPGPU	general-purpose GPU
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HACD	Hierarchical Approximate Convex Decomposition
HD	High Definition
HDLC	High-level Data Link Control
HLSL	High Level Shading Language
HMD	Head Mounted Display
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
ID	IDentifier
IES	Illuminating Engineering Society (Photometric file format)
IESNA	Illuminating Engineering Society of North America
IFC	Industry Foundation Classes
IGES	Initial Graphics Exchange Specification
IO	Input/Output
IP	Internet Protocol
ISO	International Organization Standardization
IT	Information Technology
JSON	JavaScript Object Notation
JT	Jupiter Tessellation format
LED	Light-Emitting Diode
LOD	Level Of Detail
MCAD	Mechanical CAD
MPFR	Multiple-Precision Floating-point computations with correct Rounding library
NX	NX Siemens PLM Software
OBJ	Wavefront File Format
OCE	Open CASCADE Technology
OPC UA	Open Platform Communications Unified Architecture
OSG	OpenSG
OSM	OpenStreetMap
OWL	Web Ontology Language

PC	Personal Computer
PDF	Portable Document Format
PDM	Product Data Management
PET	Positron Emission Tomography
PLM	Product Lifecycle Management
PLY	PoLYgon File Format
PS	Power Supply
RDF	Resource Description Framework
REST	Representational State Transfer
ROS	Robot Operating System
RPM	Revolutions Per Minute
SHP	SHaPefile format
SKP	SKetchuP file format
SLAM	Simultaneous Localization And Mapping
SME	Small and Medium-sized Enterprises
SSH	Secure SHell
STEP	Standard for the Exchange of Product model data
STL	STereoLithography
TIFF	Tagged Image File Format
UDP	User Datagram Protocol
UI	User Interface
UST	Ultra Short Throw (projector)
UV	UV texture coordinates
VB	Visual Basic
VE	Virtual Environment
VMU	Virtual Mock-Up
VR	Virtual Reality
VRML	Virtual Reality Modeling Language
VRPN	Virtual-Reality Peripheral Network
VTK	Visualization Toolkit
XML	eXtensible Markup Language

1 Introduction

Virtual reality (VR) is the ultimate technology to experience simulated environments. VR applications allow to share dreams and visions as if they were real. Virtual environments can be used to explore abstract scenes or realistic worlds. Computer games do successfully make complex interactive real-time applications since many years, and extending towards more immersive hardware is a natural development happening right now. In sciences and engineering the use of virtual reality has been established for a little longer. Virtual environments are especially interesting for engineers and scientists to transform their abstract data into an intuitive representation. Interactive and immersive environments allow to experience virtual scenes in a manner as close to reality as possible. Scientific data or math constructs can be brought into context, making them easier to analyse and communicate to others. Engineering problems and solutions greatly profit from virtual testing and validation applications. Realistic representations of humans, machines and environments allow for example to create planning, validation or training applications. This kind of technology has been predicted in many science fiction works in ways that go far beyond the current state of the art, but nonetheless show that virtual reality has the potential to impact any aspect of human life. It may even induce the next big technological revolution, a "virtual revolution" in industry and society. The underlying fundamental paradigm to this is to put the user in the center, to consider the user as a "resourceful human" instead of a "human resource" [OHH+15, HVH+13, LBD+18].

Virtual reality applications are mostly defined as having a real-time and interactive application logic as well as being deployed in immersive hardware setups. Virtual engineering [Ovt10] applications combine this further with real-time simulations and various interfaces and tools with the purpose of front-loading and complexity reduction throughout the whole product life cycle. In the product conception and design phases, virtual reality can be used to experience the product design as close to a physical mock-up as possible, even basic functionality can be expressed as simple animations. This has the advantage to include non experts into the product design conception from the get go. This can for example help to integrate the customer more efficiently in the communication process and incorporate his wishes, requirements and comments more accurately [BMKSH09a, BMKSH09b, KHO13, KHO15]. The importance and potentials of continuous and individual customer integration into the product development process is shown in [LB17, BBA+10, BBAO11, ZBO12, BO13]. Another example is the strength of immersive representations for enhancing the communication in the product development pro-

cess [EHOO11, HBO13, HO16]. Virtual environments can also be useful for cross-domain and cross-enterprise collaborations [MSO07]. Data can be integrated from sources that are spread across continents into a single collaboration space, allowing engineers to work together as if they were in the same room.

Product design and development is in need of advanced collaboration environments and new paradigms due to an ever growing complexity [WBO15, WBH+16]. Virtual reality is an important part of the whole product life cycle management [SNO14, NO13, NBO13]. Some work analyses the possibilities to interface between PLM systems and VR environments [HSO08].

After the product development phase, comes the production planning, monitoring and optimisation phases. The virtual factory is a major field where virtual reality definitely has its place [GPC+07, KSO+07b, KSO07a, RO09, RWO12, ORK+08, SKAO+08, SOW09, SWA+10, HSHR13]. An important aspect of manufacturing is simulation, especially processing simulations. CAE methods like FEM can produce huge data sets that can be incorporated into virtual environments [SHO14, HSW+14]. Physical properties like radiation or magnetisation can then be explored by all the sensory channels addressed by the immersive hardware setup [MBC+12]. The production planning can also impact the product design [WBO11].

Even though virtual reality concepts have been around for decades, and the gaming industry is hyping that technology with moderate success, it is still hardly common. This work aims at developing a stepping stone towards a broader deployment of virtual reality applications in all areas of science and engineering. The approach taken is to build up a virtual reality authoring tool, a software package aimed at easing the creation of virtual worlds, and the deployment of those worlds in advanced immersive hardware environments like distributed visualization systems. There are major challenges to overcome. When confronting small and medium enterprises with virtual engineering concepts, one has to face the harsh reality of numbers, especially the investment costs against the added value for the company. To create a substantial incentive for such companies to invest, one has to provide a solution to the following challenges:

- How to access and integrate the product development data of the company. Typical plant engineering is for example split in primarily three to four domains. Mechanical CAD contains the geometric models, electrical CAD the wiring and the automation data the programming of the PLCs. In some cases, building information model (BIM) data is also necessary as machines may have to interface with building resources.

- How to integrate VR efficiently in the engineering processes and workflows. Each domain has its ecosystem of CAD software tools and specialised engineers that model the product data independently of other domains.

- Show a definitive added value derived from the features for interacting, exploring and validating the engineering progress. Preventing major losses by improving the early detection

of errors and design flaws is not easily translated into potential savings like optimizing a production line.

- Integrate interactive simulations, configuration logic and much more to diversify the use cases. Once fully virtualized, the model can be used for many different applications, far beyond validation of the construction process. Training applications for example have a huge potential. Based on the virtual model, usage, monitoring or even maintenance can be trained by combining the model with a tutoring system.

A method is needed to integrate the CAD data from the product development process to create a fully functional virtual model. A software solution that addresses the challenges above has to overcome technical limitations and requirements of virtual reality technologies and applications. There are many domains involved in a virtual engineering application, not only the engineering domains like MCAD, ECAD and automation but IT domains like computer graphics, IO interfaces and much more.

- Virtual reality applications have to perform in real-time, between at least 10 to optimally 60 fps. Without high frame-rates, the application is not responsive enough for interacting properly with the virtual environment.

- The classical approach to virtualisation requires a lot of time consuming, manual work. The challenge is to further optimize and automate the virtualisation. Only a fully automated approach can be used efficiently in the product development process. There is no time to spend on virtualization as the CAD development cannot be halted. When delaying the validation it becomes obsolete as it will be based on a deprecated version of the data.

- Engineering applications have to usually handle arbitrary big amounts of data, the more data can be handled by the system, the better. Even when improving the performance for a better frame-rate, the engineer will try to add even more detail to the virtual model, negating the frame-rate increase. Handling big data models is very important, even the loading process has to be efficient to reduce the duration of validation cycles.

- There is vast diversity of software tool used in the industry, most do provide poor interfaces, especially CAD systems have a poor support for neutral exchange formats, which means loosing data and accuracy. Native formats are locked behind licensing and not compatible with open source. Even without this restriction, native formats are as numerous as CAD systems, making it very difficult to support them all.

- The high costs for virtual reality software tools and the investment in know-how binds the company. Switching from one to another software tool is very difficult and costly. The reuse of tools and assets in another work-flow or solution is very difficult. Work-flows

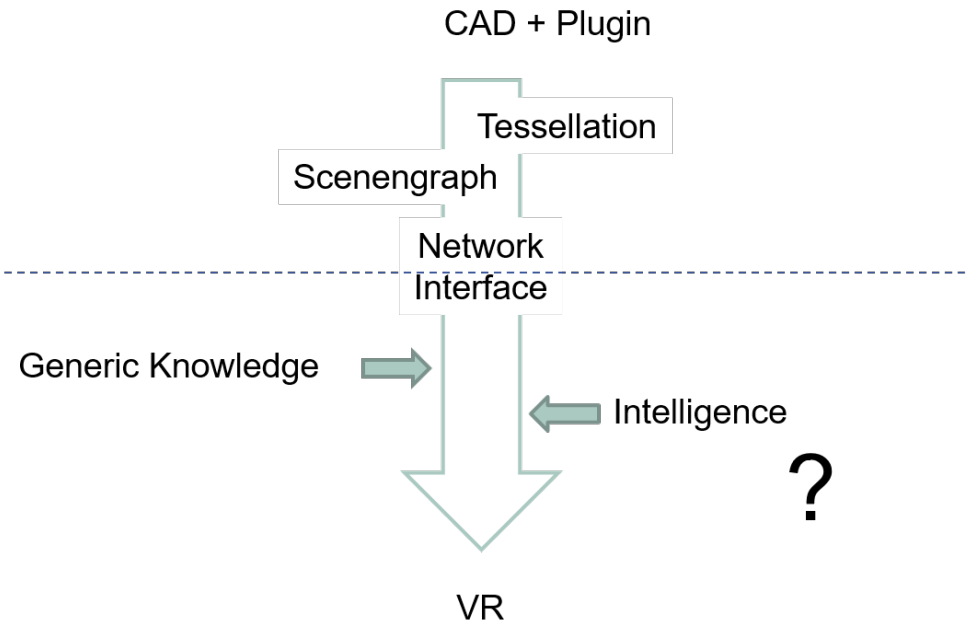


Fig. 1.1: CAD virtualisation work-flow, using a CAD plugin to achieve a bidirectional data interface and semantic modelling to automatically infuse the static data with intelligent behaviour

are very tightly encased in a specific solution concept. The challenge is to try to work closely with the CAD systems in use. Trying to change the way the company operates, for example by giving guidelines for construction engineers, will drastically limit the acceptance of virtual reality technologies.

The approach taken in this work is to develop a virtual reality authoring system to tackle the challenges described above and enable the creation for virtual engineering applications. The software has to make the development and deployment of virtual reality solutions much more efficient by further automating key aspects like the CAD virtualisation process or the hardware calibration of immersive systems. Interfaces have to be developed like support for CAD exchange formats or bidirectional communication with CAD software tools using plugins as depicted in fig. 1.1 as well as interfaces to industrial communication systems like OPC UA. Such a system will also have to address reuse and automation of application logic. For this it is necessary to handle the enrichment of geometric data with semantic information to reduce the amount of manual work like scripting the application logic, as well as to integrate heterogeneous data as depicted in fig. 1.2.

1.1 Research Problems

This thesis is situated in applied engineering sciences with a heavy focus on software development. It is important to define the research aspects, the open questions that require a methodo-

logical approach to be answered as well as the core innovation of the system conception. The challenges described above can be summarized in the following research issues.

1.1.1 Virtualization

The first and most basic problem is the slow and tedious as well as resource intensive creation of virtual environments. There is a huge potential for engineering applications, but authoring virtual environments has to become much faster and more accessible, especially to users without much expertise and experience in application development. For virtual engineering applications it is even necessary to provide IO interfaces to fuse heterogeneous CAD data into a consistent virtual scene as described above. This has to be solved in order to create fully functional, intelligent and interactive models. Building on an automated virtualization workflow, how intelligent can a system react to user interactions and how can such a system be built at all. Solving this is a first step to replace the manual scripting of application logic.

1.1.2 Modelling Intelligence in Virtual Environments

The second problem is the lack of semantic modelling or the overall lack of semantic information in CAD data. Without semantic information, a virtual scene is only available to be interpreted by the user, the machine is not aware of much more than a collection of triangle data without explicit knowledge about the meaning of each object. A solution concept has to address how to

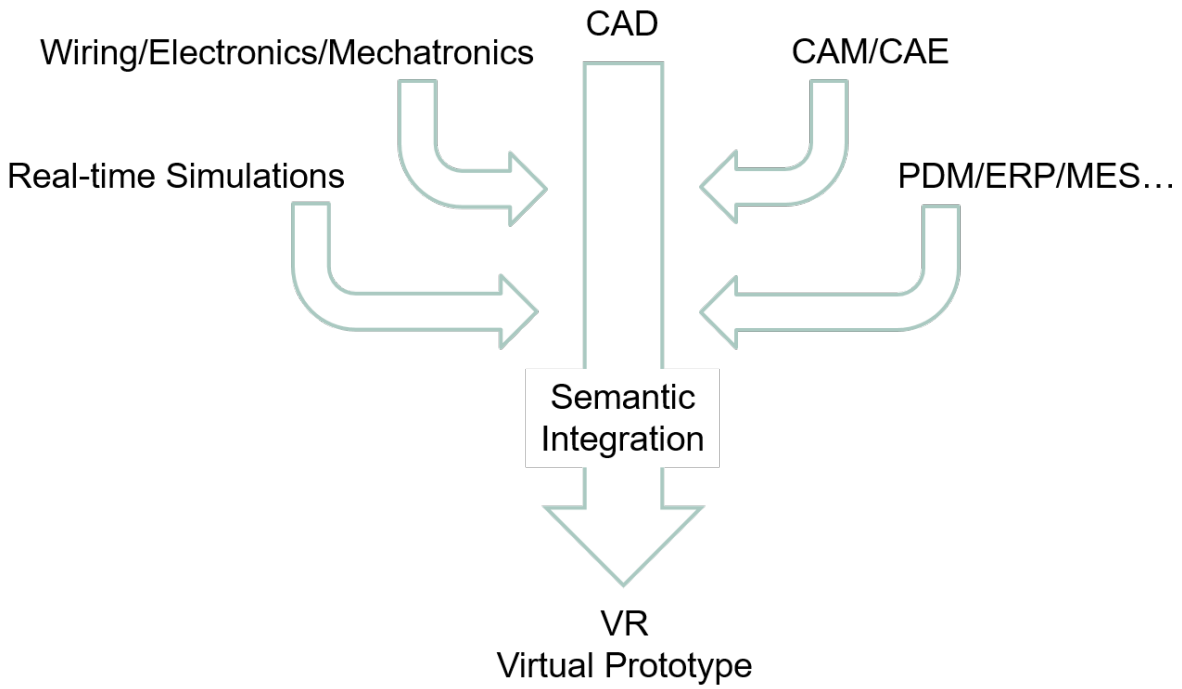


Fig. 1.2: Virtual prototype virtualisation work-flow, in addition to CAD data, the wiring and ECU programming is needed to build up a fully dynamic and functional virtual prototype

model the semantic layer of the virtual scene, and how to automate it for production workflows in industrial applications like design reviews. Solving this will aid in the virtualization processes as well as enable reasoning systems to traverse the semantic layer. This opens up to many advanced applications that can use semantically enriched virtual environments to offer the user a much richer experience. An example are training applications where tutoring systems can use the explicit knowledge of the virtual environment to evaluate the users actions and generate new tasks based on the available assets.

1.1.3 Deploying Virtual Engineering Methods

This research problem is about the difficulties to deploy virtual reality solutions and work-flows in industry. This is especially difficult for the product development process as the requirements of enterprises can vary and are often quite strict. The first aspect is to analyse the software landscape of the company as well as the information explicitly modeled in CAD construction data. This defines how much the virtualization process can be automated at all and dictates which data interfaces are needed to setup the virtualization work-flow. The second aspect is to integrate the work-flow in the product development and design review processes of the company. It is also important to bring the engineers on board and motivate them to use the system productively. It is often necessary to have proponents of virtual reality technologies in order to successfully deploy new workflows in SMEs. Those can be team manager or head of department. The system design has to consider and avoid possible issues that might reduce acceptance.

1.2 Research Questions

Now it is finally time to define research questions. The research problem in this work revolves around identifying and reducing obstacles for using advanced virtual reality technologies in engineering, finding new approaches to create added-value with virtual environments, raising acceptance in industry, and thus overall enabling scientists and engineers to author virtual reality applications. This leads to the following research questions, subdivided into scientific research questions and practical research questions. While the scientific aspects are intended to cover more methodical issues like the authoring or deployment processes, the practical aspects cover issues like the software implementation and different industry use-cases. The research questions addressed in this work are:

1. How to increase efficiency of virtual environment authoring for engineering applications?
    - How to optimize the design of virtual reality systems?
    - How to facilitate the creation of rich interactive virtual worlds?
2. How to reduce the amount of work for open world virtual environments?
    - How to generate open world assets like road networks or buildings?
    - How to build semantically rich environments for intelligent applications?
3. How to automate the virtualization process for virtual engineering applications?
    - How can MCAD, ECAD, Automation data and BIM data be fused into a consistent functional virtual model?
    - How can virtual engineering solutions be deployed in SMEs

1.3 Summary

This thesis is about opening up virtual reality technologies for engineering applications. Virtual environments are the closest to real environments without physical limitations. They are the perfect space for experiencing, testing and validating during a process of research or engineering. The challenge is that creating interactive real-time applications requires a broad knowledge of application development, computer graphics and much more. There are many software tools that try to ease the development of such applications but they are mostly in the gaming domain, lacking advanced features for interfacing to CAD systems, interfacing to industrial communication interfaces or virtual reality capabilities like clustering or tracking. This is the first focus of this thesis, creating a software system that greatly eases the use of the most common engineering systems and allows to easily be extended.
The second focus of this thesis is to address the virtualization work-flow for virtual engineering applications, especially to support design reviews and virtual commissioning during the product development process. In short, how is it possible to use heterogeneous data like CAD, GIS or other data, fuse it together to a virtual model, construct a semantic layer on top and infuse the model with intelligence. An intelligent virtual environment in this context is a virtual scene that uses explicit knowledge to infer its behaviour and reactions to user interactions. Creating a system with such capabilities would revolutionize the use of virtual reality in industry, especially for SMEs. This would allow to finally introduce virtual engineering solutions in industry on a broad scale.

2 Theoretical Background

This chapter aims at preparing the reader to understand the basic theoretical framework, the context needed to situate the work of this thesis. The more fundamental topics are computer graphics basics as well as virtual reality. More advanced topics are the semantic web technologies and virtual engineering. The driving simulation and open world topics are use case specific.

2.1 Virtual Reality

Virtual reality is the combination of a digital 3D representation of a scene, a realistic or abstract virtual environment, and the use of immersive hardware devices to experience this virtual environment as seen on figure 2.1. In literature one can find the definition of Sherman and Craig [SC18], who describe virtual reality as:

"... a medium composed of interactive computer simulations that sense the participant's position and actions and replace or augment the feedback to one or more senses, giving the feeling of being mentally immersed or present in the simulation".

Another fundamental definition of Burdea [BC03] reads as follows:

Fig. 2.1: Virtual environment in a high-end CAVE system.

"Virtual Reality (VR) refers to a high-end user interface that involves real-time simulation and interactions through multiple sensorial channels (vision, sound, touch, smell, taste) allowing the user to be effectively immersed in a responsive virtual world".

Virtual reality is a combination of many technologies and domains to create an immersive experience. This section is rather short as most technological aspects are presented in their own domain related chapters. In the following a few key concepts are explained.

2.1.1 Presence

The term of presence is used to describe the feeling of the user when he is fully focused on the application. It is a feeling of being part of the scene, being directly involved in the flow of the application. It is important to try to avoid disrupting this feeling. If the user feels involved in the environment he can focus on the experience and interact with the environment intuitively. The intuition is a powerful tool to grasp information more quickly and understand the environment, for example complex data or user tasks, in a deeper way. Experiencing something more vividly does allow to better remember it.

2.1.2 Immersion

The term of immersion is used to describe a system, a combination of hardware and software, in regard to its capability to immerse the user in an application. It is important to make the difference with the term presence. The feeling of presence in a virtual reality system can be the result of the degree of immersion the system can offer, but the presence is purely subjective. The immersion of a hardware setup is strongly linked to the capacity to appeal to the senses of the user. This is usually done by maximizing typical hardware related aspects like:

- Maximize field of view to engage the user with the virtual environment.
- High display resolution is essential as blurred text or pixelated displays can be very disruptive to the feeling of presence.
- Provide acoustic coverage, more sound channels can allow for more complex sound design.
- Provide vibration and force feedback if possible.
- Maximize user acceptance of the system, this can mean to switch to a wireless solution, or avoid disruptive behaviour of the hardware.

It is important when considering to invest in a VR system to define minimum requirements and be cautious when configuring the system. It is not enough to maximize a few parameters like

field of view and resolution, but it is necessary to consider the system with a holistic approach. Any miscalculated detail can drastically decrease the overall immersion of the system.
Regarding software, the immersion depends on many factors. On one side there are obvious factors like rendering quality overall, light and shadows, realistic scenes with textured models and overall a well crafted virtual environment. On the other side there are much more difficult aspects that can disrupt the feeling of presence of the user. This can be a missing interaction possibility like a door that does not open, or an unforeseen user action that results in an unexpected state of the application, or simply any behaviour of an element of the virtual scene that disrupts the immersive experience of the user.

2.2 Computer Graphics

Computer graphics are at the core of any virtual reality application, especially 2D and 3D real-time computer graphics. The goal is to visualize data, compute 2D images by rendering data, projecting it on a 2D surface and displaying it using hardware display devices. 3D computer graphics handle 3D geometric data or 3D model, usually in Cartesian space, and use graphics hardware to process it into a 2D projection, often orthographic or perspective projections [Gor12]. But overall, 2D and 3D applications use the same algorithms to process vector and raster graphics.

2.2.1 Data Models

A 3D model is the mathematical representation of a 3D object, when displayed it becomes a graphic. Rendering the model allows to display it as a 2D image, but it can also be used for calculations and simulations, even for 3D printing to create a physical representation of the model. There are different mathematical descriptions of 3D models.

Polygon Mesh Model

Mesh models are the most basic kind of 3D data. The mathematical model of polygon mesh formats is based on vertices and the OpenGL primitives, points, lines, triangles and quads [Gor12]. A Vertex is a position in 3D space, along with other information like a surface normal, color or texture coordinates. The mesh is a set of vertices and a list of indices that define the OpenGL primitives, the faces of the mesh. An example is depicted on fig. 2.2. A mesh is very easy for an application to handle, there is no ambiguity on how to interpret the data. Those can also directly be transferred to the graphics card and visualized without having to process the data in any kind. Meshes can also serve other purposes like using it for collision engines in real-time applications or ray casting the mesh to interact with the model, like selecting a face or vertex.

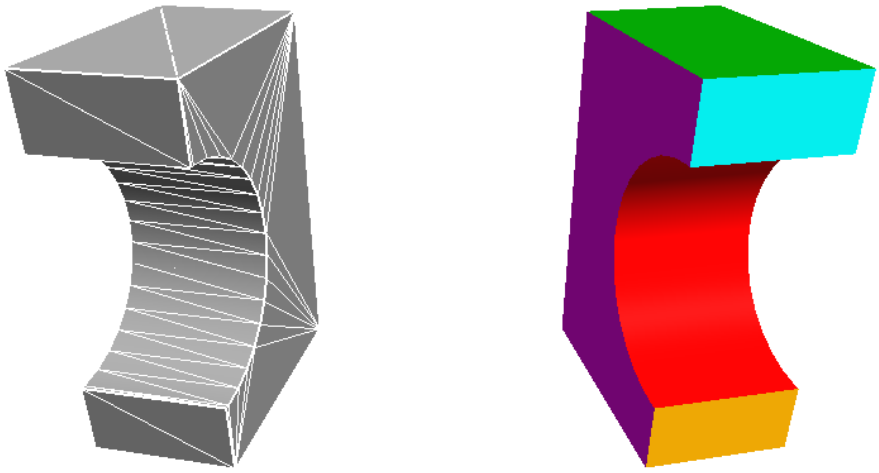


Fig. 2.2: Model as polygon mesh (left) and BREP (right)

Boundary Representation Model

The boundary representation model is much more complex than the mesh model. It is mainly created using Computer Aided Design software, for example for machine construction or building design. The geometry is described using analytic descriptions of surfaces and bounds instead of points and polygons [Str06]. This allows on one hand to exactly model curved surfaces while avoiding precision loss due to tessellation accuracy, and on the other hand it contains higher level semantic information, useful for example for consistency checks or simulation models. An example is depicted on fig. 2.2. The colored faces indicate the different mathematical surfaces the shape is constructed with.

Volumetric Data

Volumetric data is any kind of data using a space partitioning data model. This is typical for numerical simulations like flow simulations or medical data like PET scans [LSPV15]. The simplest form is a homogeneously discretized voxel field. This kind of data can also be transferred to the graphics card as a 3D texture, used for example with procedural generated virtual scenes or ray casting visualization tools. For real-time applications this kind of data is often reduced to a mesh using specialized algorithms. This greatly simplifies the handling and visualization of volumetric data.

Point Cloud

An interesting case of 3D data are point clouds. A point cloud is essentially a mesh without faces, just a set of vertices [LW85]. This is especially interesting when dealing with huge data sets with hundreds of millions of points, multiple gigabytes per file. Such huge point clouds are easily generated with current state of the art scanning hardware. Some 3D scanners also

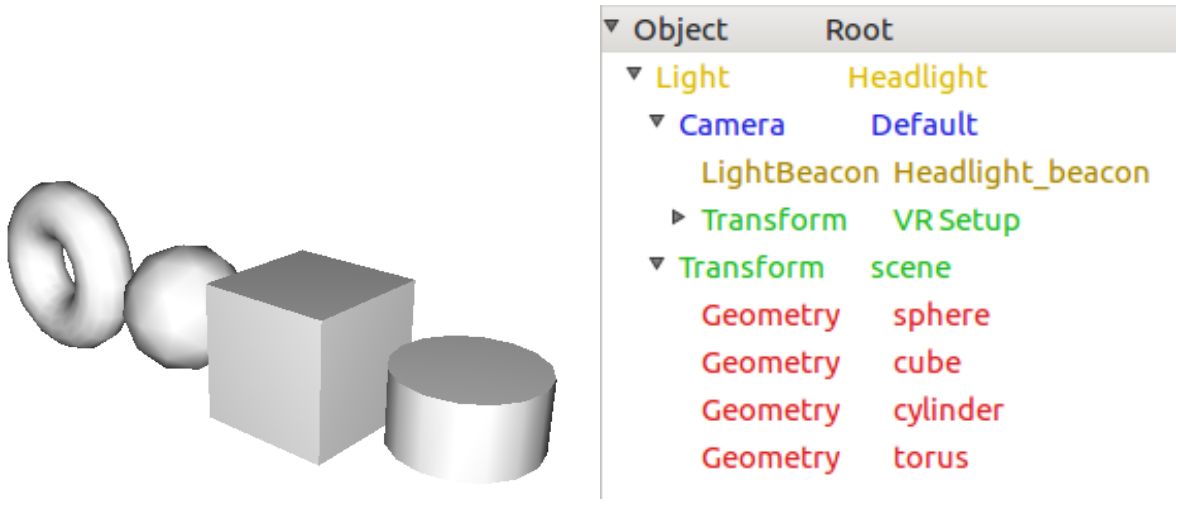
provide color information for each vertex, allowing quite realistic visualizations of point clouds. Point clouds are often greatly simplified by decimation of the set of vertices and triangulation, obtaining a mesh for display purposes.

3D Exchange Formats

It is very important for an application to handle those different data models correctly. This is why standardized exchange formats are employed to transmit data from one application to the other [DV11]. There are two major types of exchange formats for 3D geometric models, mesh formats and BREP formats. Mesh formats are very easy to serialize, but can become very storage consuming. On the other hand, the analytic geometry descriptions are much more complex to store, but also more compact than big amount of triangles needed to approximate the geometry, especially for curved surfaces.
Independent of the type of geometric model, some formats can store additional meta information. The most important is scene graph information to store complete scenes. More on scene graphs in the next section. Other meta information may be product data, object classification or identification.

2.2.2 Scene Graph

A scene graph is a hierarchical graph structure whose nodes are the components of a virtual scene and the edges are parent child relations. This structure represents the topology of the scene, how objects are clustered together. The scene graph structures the scene elements like geometries, but also specialized nodes like for example camera and light nodes [HVDF+14]. Figure 2.3 shows a simple example of such a scene graph structure. To work on the scene



(a) (b)

Fig. 2.3: a) Simple scene, and b) the corresponding scene graph.

data, an action function can traverse the scene-graph. This is useful for the rendering or for ray casting.

The basic node types are the group node, the transform node and the geometry node. The group node is an empty node that only structures the scene-graph, managing its child nodes. Every other node inherits from the group node. The next node type is the transform node, it defines a new coordinate system, a position and orientation in space. The last basic node is the geometry node, it inherits from the transform node, thus it has a position and orientation in space, but it also contains a mesh. The placement of the mesh in the global scene is defined by its local transformation, but also its position in the scene-graph. The global transformation of the mesh corresponds to the combination of the chain of transformations of its parent nodes up to the root node of the scene. This means that if two objects should stay together in a common frame of reference, then they have to be grouped under the same parent node. A simple example is a table with a fork, plate and knife. The cutlery should be grouped under the table node, which itself would be grouped under the room node. This allows to move the table around in the room without the things on the table floating in mid air, instead they will remain on the table.

The most important specialized nodes are the camera and light nodes. The camera node adds a viewpoint in the virtual scene. It is essentially a transform node with camera parameters like the field of view and the projection parameters. The position of the camera node in the scene does not have any other effect than positioning the viewpoint in the scene. Lights actually need two nodes to function properly, the light node itself defines the parameters of a light source like the light type, intensity or color, and an additional transform node defines the position of the light in the scene. The position of the light node in the scene-graph defines what portion of the scene is actually lit by the light source. Only its sub-tree, its child nodes and their descendants, is affected by the light.

There are many operations, actions and manipulations that can be applied to the scene graph. The most important operation is the rendering action. In every frame, the action traverses the scene graph and computes the transformation of the active camera viewpoint, the world transformations of each object, their visibility, the active light sources and anything else relevant to prepare the rendering of the scene. All that information is used to set the rendering state of the OpenGL context. Optimization algorithms on top of the basic render action may for example sort the objects depending on their material to avoid unnecessary communication overhead with the graphics card. This is important because every change of the rendering state has to be communicated to the graphics card, which can quickly become a bottle neck, impacting the overall frame rate of the application. Another typical operation on the scene-graph is ray casting. This is the basic tool for interacting with objects in the scene, needed for example to compute at what the user is currently aiming with his interaction device. The ray cast action traverses the scene-graph, descending along nodes that are intersected by the ray. Once a geometry is found, the operation uses the polygon mesh data to find the triangle hit by the ray, the hit position in

world coordinates and other data like hit normal or UV coordinates. Another kind of operation is the manipulation of the scene-graph structure. This is used for example for the drag and drop functionality. When interacting with an object and executing a drag event, the targeted object is removed from its old parent node in the scene-graph. It is then appended to the node corresponding to the interaction device. By moving the device, thus changing the transformation of the device node, the dragged object is moved around. When triggering the drop event, the inverse operation is applied to the scene-graph. The dragged object is removed from the device node and put back under its original parent node.

2.2.3 Materials

An object has geometric information, its shape, but this alone is usually not enough to visualize it. A material is a description of the appearance, the properties of the surface of an object. When rendering an object, the surface polygons are projected in 2D. Then the lightning information and the material information is combined to compute the resulting color on the screen. The rendering is described in more details in the following section. The material information defines for example the color components of the surface like the diffuse color as well as the ambient and specular colors. When using textured models, the textures are attached to the material. Another important function of materials is to handle shaders, code fragments that are directly injected in the rendering pipeline (see next section). Furthermore a material defines many OpenGL flags that can impact the way a texture is applied, how a wire representation is rendered or how the depth buffer is written to.

Basic Lightning Model

A lightning model defines how to compute the color of a surface fragment. The most common lightning model is to combine three color components into the final color, the diffuse component, the specular component and the ambient component.

$$
\begin{aligned}
diffuse &= (\vec{N} \cdot \vec{L}) \cdot material.diffuse \cdot light.diffuse \\
specular &= (\vec{R} \cdot \vec{L}) \cdot material.specular \cdot light.specular \\
ambient &= material.ambient \cdot light.ambient \\
color &= diffuse + specular + ambient
\end{aligned}
$$

With $\vec{N}$ the surface normal, $\vec{L}$ the vector to the light source and $\vec{R}$ the reflected eye vector. This model is not physically accurate, but gives enough artistic freedom to easily describe most materials and effects. The top row of shapes on figure 2.4, from left to right, shows the material with only the diffuse component, then with the specular component and last with an ambient component. To add surface structure or additional detail to the shape one can add a texture to the material (blue shape of the mid row on figure 2.4). A texture is a pixel image, a 2D or 3D grid

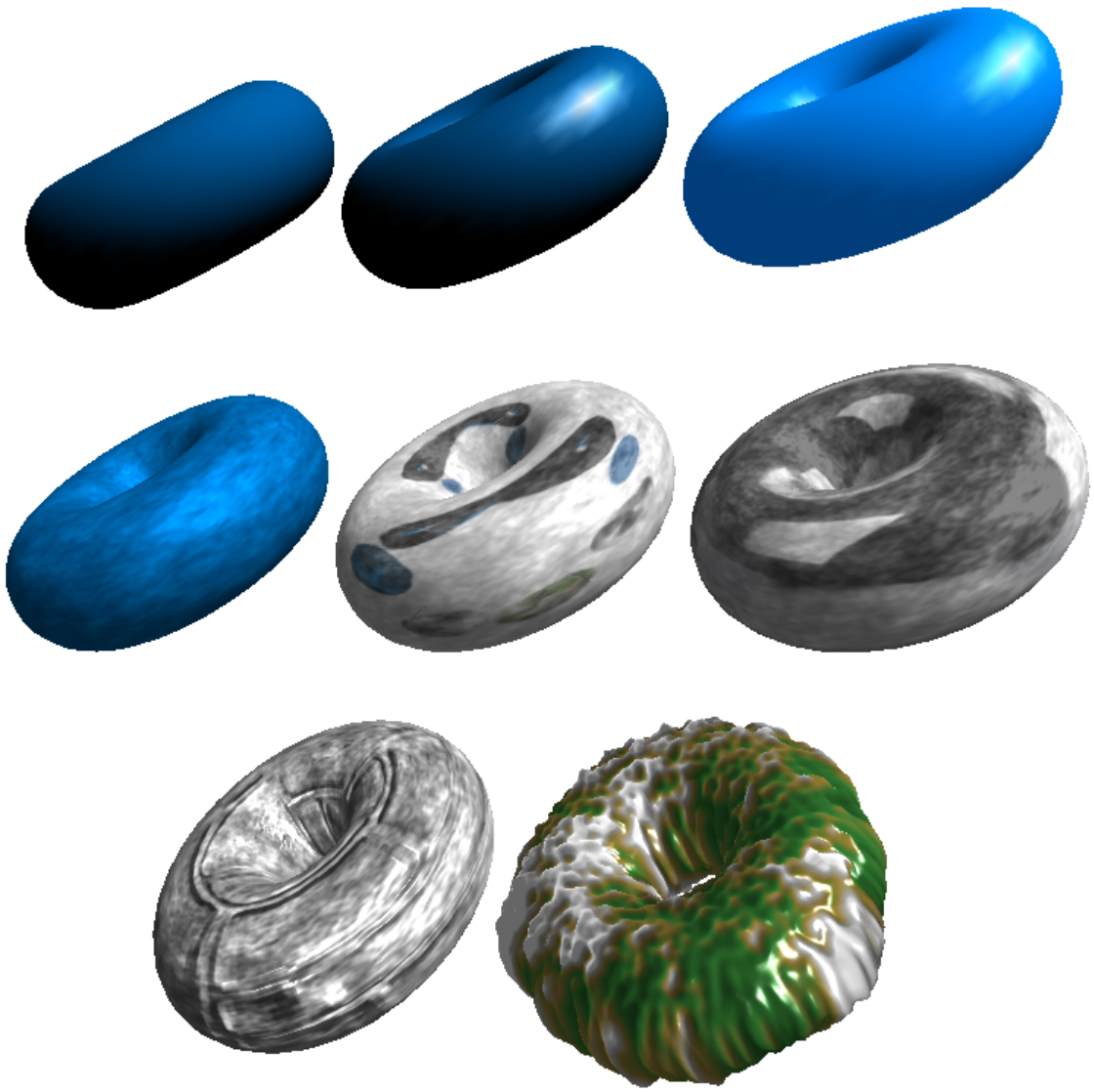
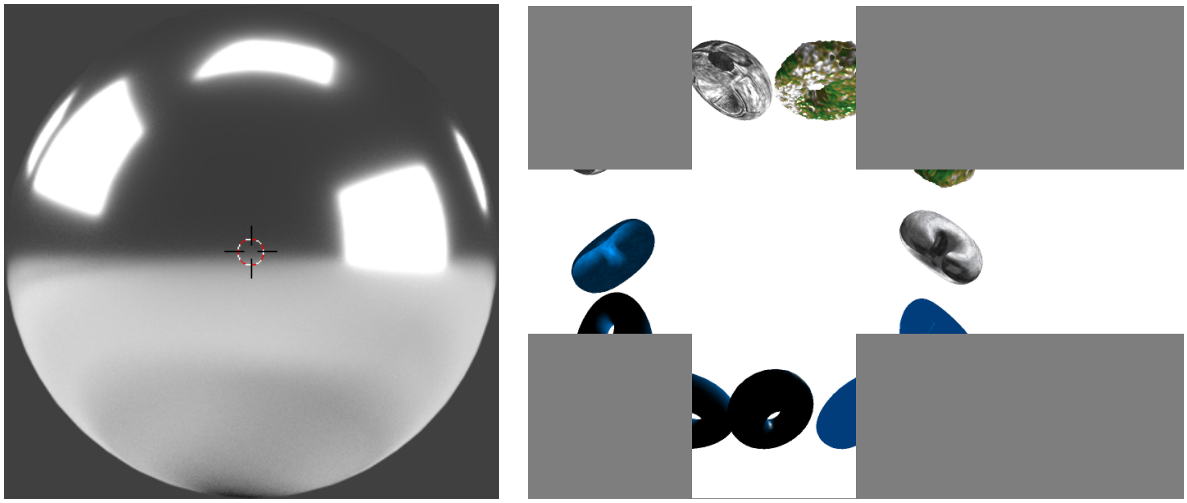


Fig. 2.4: Various surface materials. Top row, simple colored surfaces: diffuse, diffuse with specular, diffuse with specular and ambient. Mid row: textured material, reflective material, metallic material. On the last row a material with bump mapping and one with displacement mapping, both realized with GLSL shaders.

of colors. The data is attached to the material and used when rendering to modify the diffuse material component. To apply the texture to the surface fragment it is necessary to access the image color using texture coordinates. Those coordinates can be attached to vertex data of the geometry, or generated using build-in OpenGL texture coordinate generation algorithms.

Reflective Materials

An advanced surface material effect is the reflection of the environment. this can be a subtle effect that can be used for glossy surfaces, more pronounced for metallic surfaces or a clear reflection of a mirror surface. The first possibility to get an simple reflection effect is to use



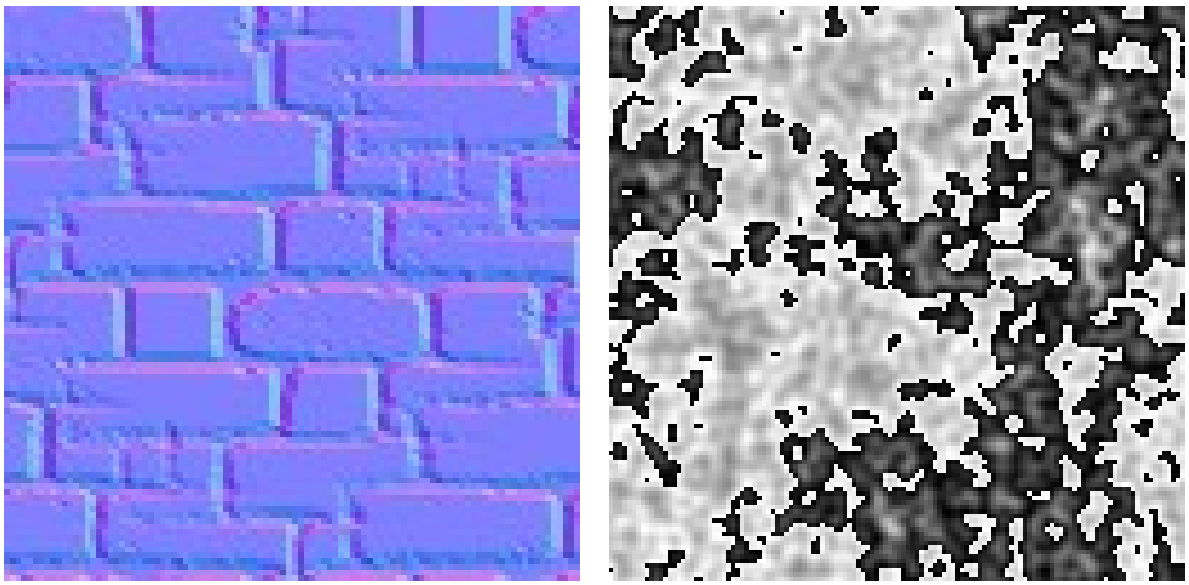
(a) (b)

Fig. 2.5: a) Spherical environment map and b) Cubemap used for the materials shown on fig. 2.4

spherical environment map as depicted on figure 2.5 a). The spherical environment mapping is not accurate enough for correct reflections, but it is computationally inexpensive and easy to implement. It is enough for creating a glossy or metallic look. To represent correct reflections it is necessary to provide a cube map and use it to render the reflection effect on the shape surface. An example is shown on figure 2.4 where the central shape has a material that reflects the surrounding shapes. The cube map that has been generated for the reflection is shown on figure 2.5 b).

Bump and Displacement Mapping

A special usage of the surface material is to add effects that change the appearance of the geometric properties of a shape. Bump mapping for instance allows to change the surface normal during the rendering process. The surface normal is tilted for each fragment using the information encoded in a special texture, the bump map.
The last material effect represented in this chapter is the displacement map. This effect is similar to the bump mapping, but instead of modifying the surface normal the surface itself is distorted by moving it along the surface normal vector. As for the bump map, the distortion is encoded in a texture, a so called displacement map or sometimes a height map, especially for generating topography.
An example for a material using bump mapping is depicted on figure 2.4, last row on the left. Next to it is an example for a material using displacement mapping. The textures used for the examples are shown on figure 2.6



(a) (b)

Fig. 2.6: a) Bump map and b) Displacement map used for the materials shown on fig. 2.4

2.2.4 Rendering

To render the scene to a display, a render action traverses the scene graph and executes the rendering pipeline for every object [HH+18]. The OpenGL rendering process of an object is depicted on figure 2.7. The rendering pipeline has different steps, some are fixed, built-in functionality, and some can be exchanged with custom code, so called shader programs. There are four types of shader, the vertex and fragment shader, the tessellation and the geometry shader [SSKLK13]. The vertex shader is executed for each vertex, usually to transform the vertex position into camera coordinates. The fragment shader is executed for each fragment, a fragment is similar to a pixel, it is not yet called a pixel because it might be occluded and not visible on the final rendering result. The fragment shader usually outputs the color of the fragment, taking into account the lightning and surface properties. For example textures are applied in the fragment shader, as well as lightning calculations like Gouraud or Phong shading. The tessellation shader is an advanced shader, not every graphics card does support them. The use is to subdivide polygons, a typical example is to use it in combination with displacement maps to render topography in an efficient manner. The last shader, the geometry shader, allows to create additional geometry based on the input primitives, for example if the object is a point cloud, then a geometry shader can be used to replace each point by a quad and place a texture on it. This can be useful when rendering the particles of a particle simulation. All those shaders are written in GLSL, a scripting language, that looks similar to C. In the context of this thesis, it is very important for the development of real-time graphical applications to be able to script in GLSL and apply the shader programs to the material of a virtual object.

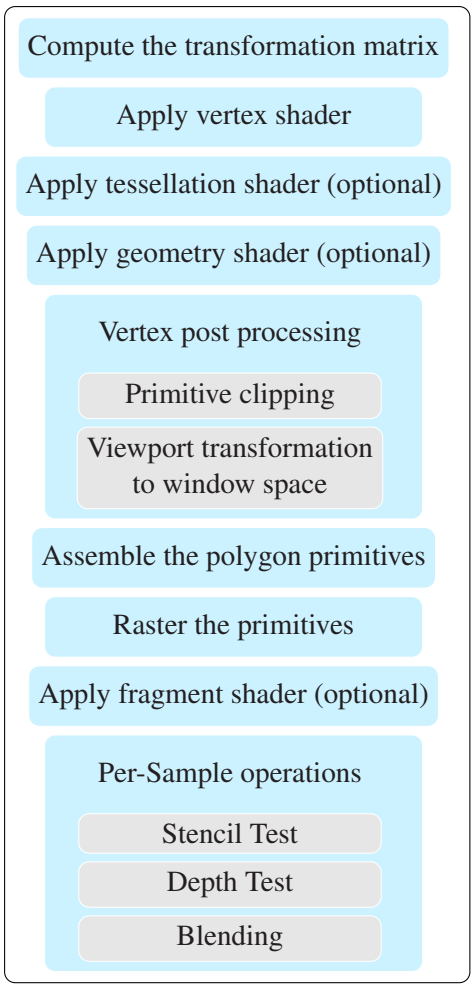


Fig. 2.7: OpenGL Rendering Pipeline

A more advanced rendering concept is to split the single rendering process into two passes. This is called deferred shading, because the lightning computation is done in the second pass. During the first pass, the rendering action traverses the scene-graph and renders each object in three buffers, the position, normal and color buffer. As this step does not compute the shading or any lightning calculation, it is much more efficient and faster. The next passes do not require a traversal of the scene graph, they can directly work in image space on the three buffers, positions, normals and diffuse colors. This means that the shading computation, done in screen space, is much more efficient and does allow to place much more lights in the virtual scene than the classical rendering process.

2.3 Sound Synthesis

A virtual environment is mainly a visual experience, but sound can be very important too for various use-cases. The simple method of incorporating sound in a virtual environment is to load sound assets, for example by loading a common file format like mp3. For more advanced use-cases it might be necessary to derive the sound from the state of the scene or any dynamic parameters. Sound is generally speaking the wave functions emitted from the speakers of the virtual reality hardware setup. To compute those functions is called sound synthesis. A typical hardware installation will have 1 to 8 channels. For each channel the sound function has to be computed and stored as discrete values in a sound package, a buffer with floating point values. The size of the buffer is equal to the sample rate and the package duration. The sample rate is the resolution of the time discretization, whereas the package duration is the duration that the sound will play once the sound card sends it to the speaker.
The first rule to follow when synthesizing sound is to guarantee the continuation of the first order derivative of the function, especially important for the beginning of a new package. If this is not given then one will hear sound artifacts.
There are different types of synthesis, the first one is the FM synthesis. A very basic method which simply computes the wave function using a carrier frequency and amplitude and a modulating frequency and amplitude. Another type of sound synthesis is to Fourier transform a spectrum of frequencies.
When changing the input parameters to the synthesis algorithm, then the resulting wave will have to be interpolated to allow for a continuous change of the sound. It is not possible to interpolate two sinus waves naively by adding and dividing, this results in a strange behaviour of the resulting sound function. The way to go is to use a so called phasor. A phasor is a complex number, which generates a sinusoidal wave when applying the time propagation operator on it. Given the wave W with an amplitude A, angular frequency f as well as a phasor $P(0) = a + i \cdot b$. The generated sinusoidal corresponds to 2.1.

$$W(t) = A \cdot \Im P(t) \tag{2.1}$$

Propagating the phasor over a time $T = t - t_{-1}$ corresponds to 2.2

$$P(t) = e^{i \cdot T \cdot f} \cdot P(t_{-1}) \tag{2.2}$$

When interpolating between two frequencies or amplitudes, then those can easily be linear interpolated and plugged into the equations above.
Some examples where dynamic sound synthesis is useful are for example an engine sound depending on the engine rpm, or the sound of virtual interactive instruments.

2.4 Semantic Web Technologies

The Semantic Web is an extension of the World Wide Web as defined by the W3C [BLHL01]. It defines common data formats and exchange protocols over the web. The most important one is the Resource Description Framework (RDF). The Semantic Web provides a common framework to share data and reuse it across applications. It is therefore a web of data, across different content, applications and systems. One important aspect is that machines can process it, the meaning of the data and its topology is machine-readable. This should enable computers to become capable of analyzing all the data on this "semantic web". But this web is still mostly theoretical, far from being used on a significant scale.
Even without the global semantic web, the semantic web technologies already have been employed in many domains, especially for engineering applications and data analytics. The main usage is to create data sets with structured knowledge, in a human and machine readable format. This, combined with an inference logic system called reasoner, allows a whole different approach to artificial intelligence.

2.4.1 Ontologies

An ontology in the context of the semantic web is threefold, it contains a taxonomy, a set of inference rules and a set of entities. We distinguish in between generic ontologies that contain only the taxonomy and the set of rules, and a specific ontology that also contains entities. The taxonomy is a hierarchical structure where the nodes are generic concepts and each edge is an inheritance relationship. Each concept has a set of properties, differentiated as data properties like integer, float or string data types, and object properties, which are properties whose types are other concepts. The entities are instances of the concepts defined in the taxonomy. They represent the concrete semantic knowledge of a small part of a world, real or fictional, or an abstract system. The ontology is the data model used by the reasoning system, described in the following.

2.4.2 Reasoning

The reasoning engine is a system that, based on an ontology, can process queries. It can use rules to make inferences, choose courses of action and answer questions. It does so by traversing all available data, usually ontologies, and using logical reasoning. The logic used by the inference engine is based on the if-then type of rule. This is a very powerful general mechanism to represent logic but one that can be used efficiently with computational resources. Other, more powerful first order logic systems, quickly run into problems like working on expressions that are indeterminate or even take infinite time to terminate. The if-then expressions are also a very intuitive way to formalize knowledge.

Two mechanisms that use such expressions are forward and backward chaining. With a given rule if 'A' then 'B'. In forward chaining, the inference engine searches for any facts 'A' in the knowledge base that matches the first part of the rule, and for each fact it finds it adds the new information B to the knowledge base. The system deduces that every 'A' is also 'B'. In Backward Chaining the system would be given a goal, a question to answer. Such a question could be, is entity 'a' also 'B'? It searches through the knowledge base and determines if 'a' is 'A' and if so asserts that 'a' is also 'B'.
The inference engine works in an iterative manner. A cycle consists of three steps, matching rules, selecting rules and executing rules. The process has to be iterative because the execution of rules changes the knowledge base. The process finishes once no additional rule that matches is found. To find matching rules, the engine evaluates the available rules and checks if they can be applied with the current knowledge base. In forward chaining the engine evaluates the conditions of a rule. In backward chaining the engine evaluates the implication of rules and if they help towards processing the query. After matching relevant rules, the engine has to prioritize their execution. Then it can execute the rules and start the next iteration.

2.5 Application Authoring

This section will explain concepts relevant to the development of software applications. The focus lies on the requirements of an integrated development environment, especially the necessary developer tools and utilities.

2.5.1 Scripting

When developing an application, it is necessary to define the logic it follows, the application flow, user interactions, simulations and much more. This is done by programming, writing out in detail the behaviour of the application. This is where script languages come into play, they allow to program during the run-time of the virtual reality engine. This avoids having to stop the whole application and restart it to test it. This allows for a very interactive programming paradigm, where every change to the application logic can be directly tested.
The most common scripting language is Python, which makes it the best choice for most scripting editors. This is because, due to its popularity, most software libraries have been extended with Python bindings. It is also quite easy to extend own C/C++ code with Python bindings and import it into the scripting environment. This means that analogue to any system wide installed Python library, the custom C/C++ code will be linked through its Python bindings at run time into the Python editor environment and its functions executed in the same process as the editor. This makes it highly flexible for the user, allowing him to interface quickly with any external system. Other scripting languages do play important roles as well. There are indeed limits to scripting with Python, especially when developing web content or shader programs.

To write a shader for the rendering pipeline as described in 2.2.4, the scripting language needed is GLSL, at least when using the OpenGL API. As the shaders are executed each time the application renders its 3D scene, each change to the shader can be applied and visualized instantly. This greatly helps with developing shaders and drastically reduces development time.
To develop a website, the scripting languages are also set. The structure of a website is written in HTML, the styling in CSS and the programming in JavaScript. Web design is one of the most common tools to create 2D interactive content, and it has the advantage of being easily distributed through web browsers. This is where it gets relevant for virtual environments, as it is possible to deploy websites on textures, mapped on any 3D geometry. This is a very potent feature to ease the creation of 2D content in 3D environments like interactive panels or menus.

2.5.2 Application Flow

When designing the application logic, it is important to figure out how the execution flows, when functions are triggered, if the trigger depends on user input, or if it is time dependent. The typical triggers are:

- Application start-up, initialization phase
- Per frame execution
- Timed execution
- Signal triggered execution
  - User input
  - Network message
  - Callbacks

When such a trigger event is fired, a predefined part of the application code will be executed. This can be a whole script, or just a single function. In Python, everything is an object, even functions. This allows to easily pass functions as callbacks to other parts of the code or to a module of the VR engine. In most cases, an application will execute all the initialization scripts when starting up. The scripts that handle animations for example will be triggered with every new frame, usually with 60 Hz. Other scripts will wait for the user to trigger them when navigating through or interacting with the virtual world.

2.5.3 User Interaction

Interacting with a virtual environment is the key to experience it. When developing an application, the interaction and navigation paradigms have to be defined. In the case of virtual reality,

the more exotic hardware components than mouse and keyboard offer more possibilities. This means that when developing interaction functionality, the targeted hardware system has to be taken into account. For the development environment it is important to abstract the VR hardware configuration as much as possible from the development of the virtual environment, but also to ease the binding of interaction and navigation functions to interaction devices like button and joystick devices or tracking systems. Basic paradigms like object selection, drag and drop, fly through or orbit navigation have to be provided by the system, in a manner to easily bind to the various interaction devices.

2.6 Open World Generation

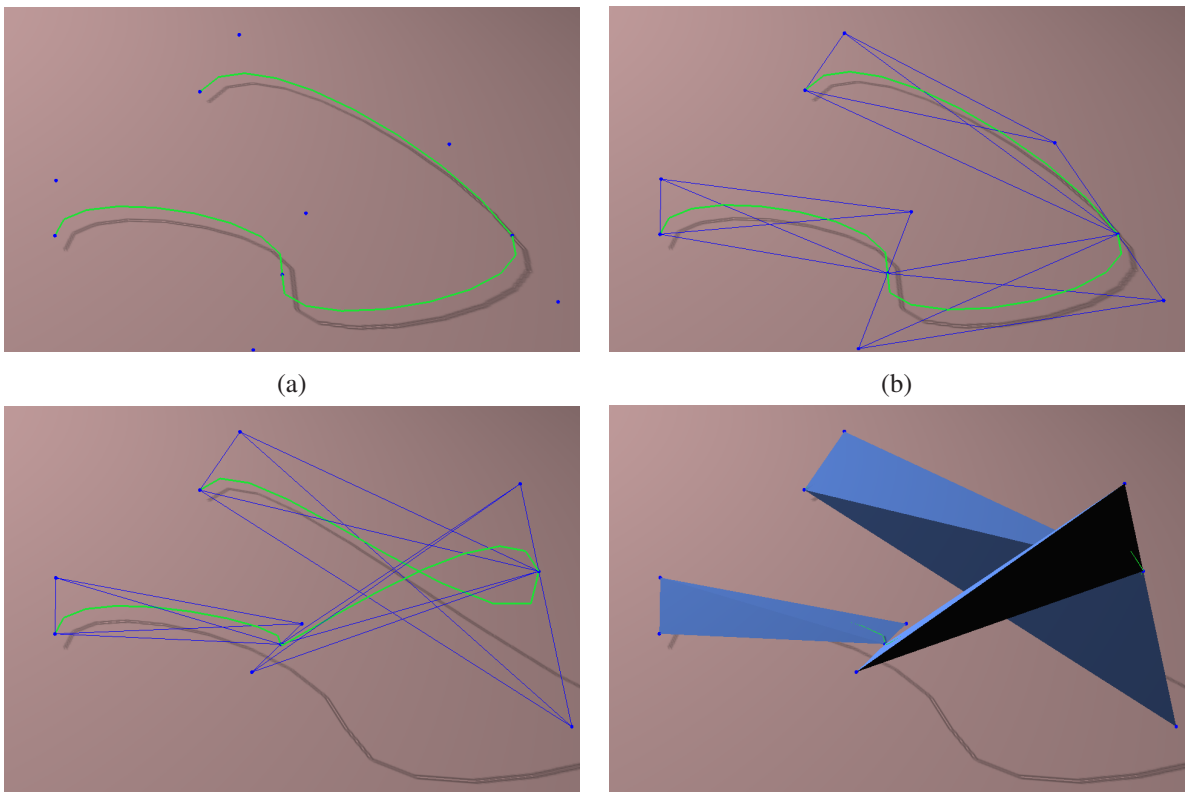
Open worlds are virtual environments that depict a world similar to ours, in the sense that it is a Cartesian space with some kind of terrain and assets. This might be a very abstract fictional world, a highly realistic representation of our world, or anything in between. This section explains a few fundamental concepts to understand how such worlds are generated, starting from the way that geographic information is stored to the generation of road networks, trees or traffic simulations. The chapter in this thesis which will reference this most is the chapter about the driving simulator implemented in the scope of this thesis, 6.1.

2.6.1 Maths Utilities

The following sections require the introduction of some basic math concepts used for the open world domain throughout this thesis.

Path Structures

For many applications it is necessary to define curved paths. This can for example be animations where rigid objects are moved along paths or generating smooth lines for abstract data visualizations. A possible way to describe complex curved paths is through cubic Bézier splines. A spline in this case is simply a set of points in space, interpolated by curves. The curvature is continuous at each point. Each point is defined by a pose in space, a position and orientation. The curves are constructed by interpolating the point positions with cubic Bézier. The orientation on any point of the curve is also interpolated. The tangential and up vectors of each point are interpolated using a quadratic Bézier curve. Additional parameters like color information can be added to the curve and interpolated for example linearly. The control points necessary for computing the cubic Bézier curve can be explicitly given or simply generated based on the points. An important property of Bézier curves is that they are limited by the hull spanned by the end points and control points as shown on fig. 2.8. This is important when for example computing the distance from a point in space to the path. The formulas for the Bézier interpolation curves are given for the linear (2.3), quadratic (2.4) and cubic (2.5) cases.



(a) (b)

(c) (d)

Fig. 2.8: Bézier spline, a) planar configuration, b) flat curve hull, c) 3D configuration, d) curve hull

$$B_L(t) = (1-t)P_0 + tP_1 \tag{2.3}$$

$$B_Q(t) = (1-t)^2P_0 + 2(1-t)tP_1 + t^2P_2 \tag{2.4}$$

$$B_C(t) = (1-t)^3P_0 + 3(1-t)^2tP_1 + 3(1-t)t^2P_2 + t^3P_3 \tag{2.5}$$

An important feature is to compute the distance from any point in space to a path curve. This can be used for example to compute collisions with the path. To compute the shortest distance it is necessary to compute the derivative (2.7) of the distance function (2.6) and search for its minima.

$$D^2(t) = (B(t) - P)^2 \tag{2.6}$$

$$D^2(t) = (B(t) - P)^2 \tag{2.7}$$

If the Bézier curve is a third degree equation, then the distance function will be of degree 6 and its derivative of degree 5.

Graph Structures

A basic graph structure is a set of nodes connected through edges. The nodes and edges are indexed for easy access. An edge can be unidirectional or bidirectional, this is important for graph-traversal actions like for example a path finding algorithm. Depending on the application it is possible to add additional information to the nodes and edges. In virtual reality applications it is often necessary to store positional information in the nodes. A position and direction allows to define Bézier curves instead of linear edges. Further additional data can be stored externally using the same indexing as the graph elements.

Space Partitioning

Space partitioning is a common technique to describe space by subdividing it following a certain algorithm. There are different algorithms, ranging from simple voxel fields over k-d trees and octrees to a combination of those on different scales.

2.6.2 Geo Information System Data

Geo Information Systems manage geographic data, usually vector or raster data. Map data for example is a type of vector data, a planar graph to represent road networks, region boundaries and punctual assets. Height maps describing the topography of a terrain are usually stored as raster data, which is essentially an image, where each pixel is a point on a even spaced raster, containing the height of the terrain at that raster point.
When handling GIS data, it is usually necessary to manage the specific coordinate systems. The data is specified in the geographic coordinate system, similar to a spherical coordinate system with omitted radius. When topography data is available, the radius component can be computed based on the height map. To efficiently manage the handling and the conversion to Cartesian coordinates, a planet management module is used. The planet modules can also manage the terrain chunks, as the planet surface has to be split into many chunks, corresponding to the size of the topographic textures. This allows to only render the chunk the user is situated on, which improves greatly the performance. It is also important to localize the global OpenGL coordinate system in the center of the terrain chunk. The reason for this is that the planetary scales will otherwise induce floating point errors in the transformations of the scene-graph, which is the source of jittery artifacts that quickly make the application unusable.

2.6.3 Topography

The terrain of a virtual environment is usually quite large, with very large features like hills and mountains, but also small scale variations. This makes it quite complex to implement in real-time applications. It is not efficient to create a giant mesh containing the whole terrain. This is where the tessellation shader comes into play. The terrain geometry can be simplified to a very rough grid, resulting in a low poly mesh. The height data is packed as a texture and attached to the terrain material. Then the tessellation shader subdivides the part of the grid close to the camera, which is very efficient as it is executed on the graphics card. The last step is to apply the height, displacing the vertices according to the values in the height map. The result is a dynamically detailed terrain, parts of the terrain far from the camera are low polygon, while the parts close to the camera are much more detailed.

2.6.4 World Asset Generation

A virtual world is filled with assets like nature, infrastructure and characters. In gaming, most virtual environments are filled with assets designed and modelled in every detail. This is one reason for the hundreds of millions of dollars for the development of triple A titles. An alternative is to generate most assets with algorithms, this greatly reduces the need for modelling assets by hand, and allows to scale the size of the virtual environment with virtually no additional costs.
A versatile tool for creating models are sweep models. Many world elements are narrow structures stretched along a linear path. This can be walls, rails, roads, fences and more. If the mathematical path is given, an algorithm can be used to extrude a profile along that path. This works well with map data, where many such assets are modelled with polylines.
Another basic method to create geometry is to tessellate polygons. This can be used to generate buildings, walls and roofs, from building outlines. There are many different algorithms to tessellate a planar polygon, but the goal is every time the same, fill up the polygon with triangles, without overlap or holes. Tessellating a polygon is often also an important step in many more complex geometry generation algorithms.
A quite advanced set of methods are the Boolean operations on meshes. The idea is well known in constructive engineering, where complex models can be modelled by combining primitive shapes using Boolean operations, so called constructive solid geometry (CSG). CSG models can be created using sweep models. This allows to easily create very complex models with a few parameters. For example, a dungeon-like model can be created with intersecting tunnels just by extruding the tunnel shapes along their paths and then subtracting them from a larger block or mountain model to obtain a complex location.
The generation of assets is not limited to the creation of geometry, but is at least as important for the creation of textures. A typical basic method, used to create many more complex textures, is the Perlin noise generator. Perlin noise is generated using multiple octaves of white

noise, multiplied with each other. The results vary depending on parameters, but usually look like clouds. Noise textures are used to add a more realistic look to materials, especially for procedural models. There are many other uses of such noise data like the distribution of assets or the distribution of biomes.

2.6.5 Driving Simulation

Driving simulators are typical mixed reality systems, somewhere between the typical virtual and augmented reality applications. The main user interface is usually more complex than for other applications, ranging from just a steering wheel to a full cockpit with pedals and gearing levers. The simulators are used in gaming and industry and come in a wide range of varieties. Apart from entertainment, driving simulations can be used for various applications for research and development or driver training. A typical hardware setup found in industry and virtual reality labs is as follows:

1. Specific user interfaces like steering wheels, pedals and gear shift levers
2. Advanced hardware components for force feedback like steering columns, suspensions or hexapods.
3. Interactive simulations, driving dynamics simulation

More advanced hardware setups can make use of automotive components, or even a whole car. The best approach to use automotive components is by connecting their built-in sensors via the controller area network (CAN). Special hardware interfaces are necessary to connect to the CAN bus. Once connected, all data passing through the CAN bus can be intercepted and analyzed, the data is usually not encrypted and thus the protocol can be reverse engineered. Using automotive components makes a driving simulator much more realistic.
The driving simulator needs a driving simulation. The mathematical simulation behind this are the driving dynamics. The simulation model consists of the three modules, the wheel and suspensions system, the engine and the gearing. The wheels are simply defined by their width and radius. They are usually attached to the chassis through a spring system with a linear constraint. The physical forces are directly applied as torques on each wheel. The engine and the gearing are described by a set of parameters, depending on the realism needed. Even complex features like stalling and the engine brake can be simulated.
An important aspect of driving simulators is the engine and car sounds as well as vibration feedback. The necessary wave data has to be generated because it is depending on the car parameters like RPM, throttle and other dynamic parameters. To generate the sound wave packages an algorithm has to interpolate between frequency spectra. A good method is to use a phasor. This is a complex number used to generate the sound wave.

2.6.6 Summary

Generating whole worlds based on algorithms is a very attractive method to create huge open virtual environments. It saves a lot of manual modelling work, and allows to easily apply major changes throughout the environment by just changing a few parameters.
An even more important benefit in generating the virtual scene, instead of using hand-modelled and textured triangle meshes, is to obtain a semantic representation of the generated world with the classification and the properties of each object, down to each triangle. This is very important for introducing intelligent behaviour in virtual environments.

2.7 Virtual Engineering

2.7.1 Maths Utilities

The following sections require the introduction of some basic math concepts used for the virtual engineering domain throughout this thesis.

Principal Component Analysis

The principal component analysis (PCA) is an algorithm that uses an orthogonal transformation to convert a set of points into a set of linearly independent vectors called principal components. The transformation is defined as to obtain the first principal component with the largest possible variance, and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components. The resulting vectors are an uncorrelated orthogonal basis set. To compute the PCA of a data set one has to first compute the covariance matrix C.

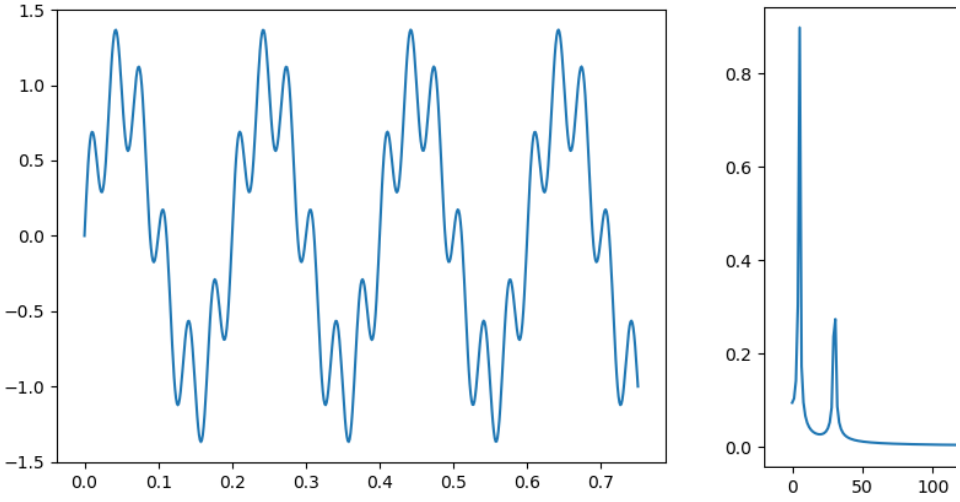
$$\vec{m} = \frac{1}{N}\sum_{n=0}^{N}\vec{p}_n \tag{2.8}$$

$$C_{ij} = \frac{1}{N}\sum_{n=0}^{N}(\vec{p}_n - \vec{m})_i \cdot (\vec{p}_n - \vec{m})_j \tag{2.9}$$

Where $\vec{m}$ is the centroid of the mesh and C_{ij} the component ij of the $3x3$ covariance matrix. Then one solves the eigenvectors of the covariance matrix. The eigenvectors $[\vec{e}_0, \vec{e}_1, \vec{e}_2]$ are defined by:

$$C \cdot \vec{e}_i = v_i \cdot \vec{e}_i \tag{2.10}$$

Those eigenvectors form an orthogonal basis and correspond to the principal components of the original data set. The PCA is an important tool to analyse geometries. Especially for computing the orientation of a geometry. This can be achieved with some knowledge of the shape, using



(a) (b)

Fig. 2.9: a) Sinus wave, b) Fourier transform showing main frequencies at 5Hz and 30Hz

symmetries to deduce which eigenvector, obtained by applying a PCA to the mesh vertices, corresponds to a semantically meaningful axis of the geometry.

Fourier Transform

The Fourier transform decomposes a function into its constituent frequencies as shown on fig. 2.9. The result is a frequency domain representation of the initial function. It is itself a function of frequency, whose magnitude represents the amount of that frequency present in the original function. The inverse Fourier transform allows to synthesizes the original function from its frequency domain representation. The Fourier transform is defined as:

$$f(y) = \int_{-\infty}^{+\infty} f(x) \cdot e^{-2\pi i x y} dx \tag{2.11}$$

The Fourier transform can for example be used to analyse the frequency of periodic features in a geometry.

2.7.2 Computer Aided Design

The main data source for any engineering application is usually a computer aided design (CAD) system, used for example to model products, plants, buildings, cars, boats or airplanes. They are mainly subdivided by the targeted engineering field in mechanical, electric and building CAD systems. This is mostly due to historical reasons as new domains came into existence, but never integrated into the older ones. Most CAD systems nowadays try to extend their functionality towards more all-rounded systems, with for example simulation packages or mechatronic

features, but they are rarely used as the engineers are specialized in their domain and use the software with its primary functionality.

Mechanical CAD

Typical MCAD software focuses on modelling consistent 3D bodies. The underlying mathematical data model is very rich in semantics, from basic geometric primitives like cubes, cylinders and cones to parametric curves, patches and volumes. The overarching structure combines these geometric bodies into CAD parts. An assembly combines parts and other assemblies as sub-assemblies. Parts can be used as multiple instances throughout the model, saving modelling time, data size and reducing complexity. To further optimize modelling, it is possible to define geometric templates. Part libraries allow to quickly reuse those templates.
A further functionality of MCAD systems consists in defining relationships between the parts to add dynamic information, constraints and much more. Kinematic chains allow to simulate the dynamic interplay between parts.

Electrical CAD

Electrical CAD software is in contrast to mechanical CAD software mostly used to create 2D plans, even if modern ECAD software also support 3D planning. Those plans represent a graph structure, electrical components that are connected with electrical or data cables. The tools allow to detail the planning down to individual sockets and pins. Even more advanced electronic components like integrated computation units as well as user interfaces like panels, monitors and other electronic components are represented in ECAD plans.

Building CAD

Traditionally, building CAD plans are 2D drawings, without any semantic information or meta data. Modern architecture systems have implemented 2.5D and 3D planning, as well as the building information model. This means that building plans will contain semantic information and meta data like the classification of building components like doors and windows and their corresponding meta data like model and manufacturer and technical parameters such as the material properties.

2.7.3 Virtual Twin

The virtual twin of a machine is its virtual representation that can be created in parallel to or after the product development process of the machine. Depending on the development stage, it can be a concept mock up or essentially the fully developed virtual prototype. It includes all dynamic and functional aspects. It can even be synchronized with the real machine to allow monitoring, change management or even system prognosis applications. The life cycle of the

virtual twin is closely linked to the machine life cycle. With the progress of the construction design, the virtual representation also grows, in size and detail, but also the amount of systems and simulations necessary to build up the virtual twin. This work describes how to build up a virtual twin with PolyVR and with particular emphasis laid on the following aspects:

- automate the data integration as much as possible
- simplify the workflows as much as possible to facilitate its deployment in SMEs

The construction of mechanical parts and machines happens in computer aided design (CAD) systems. In addition to the geometry, there can be information on the dynamic behavior of the machines like kinematic chains or animations. The CAD interfaces supported by PolyVR are described in chapter 5.5.1. The best option available for this project was the CAD tool plugin. It allows to access all geometric data, the kinematic data and product structure.

2.7.4 Virtual Mock Ups in the Concept Phase

The systematic usage of virtual models in the concept phase has many advantages. For example when meeting with a client, old projects can be loaded in the VR environment. This can help acquire new projects, show a client the progress of his machine, or discuss potential changes, wishes or issues. One can even reuse parts from archives to quickly create a mock up for a new project. For this the VR system can be connected to a PDM system or similar to access the full product data base of the company.

2.7.5 Design Review Application

The first and most basic use of immersive VR environments for CAD constructions is to perform a design review. It is the fundamental part of a virtual engineering iteration, where the current state of the project is loaded in the VR environment. The construction progress can then be analyzed with a predefined set of tools and evaluated regarding the following aspects:

- Analyze overall development progress
- Clearance analysis, check for collisions and minimal distances
- Evaluating interfaces between construction teams
- Simulating assembly to optimize production, maintenance and more

Various tools and features are needed to enable the user to explore and validate the virtual model. Those tools have to be very intuitive and flexible to foster the communication and collaboration aspects. Some typical tools are:

- Clipping plane, cuts through the geometry
- Drag and drop with snapping engine
- Undo redo system
- Visibility and transparency tool
- Selection tool and information panels

2.7.6 Project Integrator

When multiple companies are involved in a big project, then one of the companies is usually the integrator, responsible for the whole project and the commissioning in the end. This task adds another complexity to assembling the virtual twin from multiple sources, even from different CAD systems. The virtual environment can be used to plan the whole project. Even the location can be taken into account, using 2D DXF layouts or 3D scans. All features and tools described for the design review still fully apply. The project solution has to be free of collisions, including collisions with the surrounding building elements. This work proposes the design by constraints. Each machine is part of the whole material flow, thus it is a constraint to guarantee a consistent intralogistics. With the help of a process engine, it is also possible to validate different scenarios, simulate the production with its material flows and various processing steps. The interfaces between the machines should be semantically defined to automate a part of the virtualisation process.

2.7.7 Virtual Commissioning Application

The further the development, the more elaborated the virtual twin will become. At the end of the development the virtual twin has become a full fledged virtual prototype. The last step is the commissioning which usually takes two steps. First the plant has to be physically produced, mounted and commissioned. When the customer is satisfied, the whole plant has to be dismantled and deployed again in the customer facility. This process can even be delayed if errors are discovered. The whole process is very costly and contains high risks. The virtual commissioning is the attempt to use the virtual twin to simulate the whole process and thus further minimize the risks and costs. It may even one day be possible to entirely skip the first commissioning step and directly deploy the production plant at the client location.
Advanced features and simulations are needed to be able to build up a virtual prototype and successfully simulate the commissioning process. The virtual model needs dynamic and functional information. A kinematic system simulation computes the behaviour of mechanical parts. Those can be gears, chains, threads or simply rigid bodies connected with joints. The parts can form kinematic chains or graphs, the simulation has to solve such a system in real-time. The

functional information contains the programmable behaviour of the machine, usually PLCs that process sensor information, control actuators and process user input through buttons, levers or panels. If the machine has processing capabilities then numerical simulations can be used to approximate the process. To use the virtual model for commissioning the processes that will be executed on the machine have to be planned and implemented in the virtual environment. The resulting application can then be used for process optimization, validate the intralogistics and material flows or evaluate energy consumption or heat dispersion.

2.8 Summary

With all this basic introduction into computer graphics, virtual reality or application development, the reader should be able to comprehend the research problems addressed in this thesis as well as the methodology of the proposed solution and its implementation. The introduction to virtual engineering and driving simulation will be necessary to understand the validation chapter. A very important topic is the semantic web domain, reasoning and ontologies. This is important because it is a major recurring theme throughout this work.

3 State of the Art

Virtual reality is a huge field, it encompasses immersive hardware technologies, 3D interactive visualization software, interfaces to various systems and much more. Many institutes, companies and enthusiasts work with and develop virtual reality applications for a broad range of use cases.

3.1 Virtual Reality Software

To start using virtual reality technologies one needs to find a VR software tool with the necessary features depending on the application and requirements. There are many available software packages, from major gaming engines over industry targeted virtual engineering solutions to open source VR engines. This section will present some software tools for each category.

3.1.1 Gaming Engines

Popular gaming engines are widely known and have an important user base. They mainly target game developers but also allow engineers to make non gaming content and applications, especially with the addition of plugins.

Unity3D

Unity3D [Uni] is one of the most used authoring tools for creating virtual environments and games. The authoring process is separated in mostly three areas. A scene stage editor allows to configure the static scene with assets. A scripting environment allows to add the application logic. A dedicated view shows the actual game as experienced by a future player. The major workflow toggles between an edit mode that allows the developer to edit scripts or design the scene assets, and a play mode to test the actual development or game. It is possible to edit the scene during play mode, but the changes will not persist, they are thus only for testing. Scripting is an essential component of the application development, it is thus imperative for any application development environment to support it. Unity offers a built-in editor, the available languages are C# and UnityScript, a JavaScript variant, as well as Boo. Even though all those languages are supported, Boo is being dropped as it is drastically underutilized, and UnityScript is also in decline as C# is on the rise. Unity also supports writing shaders as CG code, a variant of HLSL. The code is translated internally depending on the platform. The rendering capabilities are state

of the art, high-end real-time rendering. Unity does abstract the access to the pipeline, making it easier to use but offers less control. The typical rendering passes are available like forward rendering, deferred shading and the shadow pass. Unity does provide a classical physics simulation module, needed for dynamic objects in a virtual environment, with collisions, kinematics and more. There is no separate kinematic solver or dedicated mechanics simulation though. Unity does also not provide advanced AI infrastructure, but plugins from academia are available like AiRuleBase, a forward chaining, rule-based reasoning engine. Unity can import mesh data via the most common geometry exchange formats like FBX, COLLADA, 3DS and OBJ. Unity can also read native file formats from 3D modeling software like Autodesk, Blender, Cinema4D and more by converting them to FBX. The advantage is that a change to a model will directly be imported. This is however not recommended for production code as it has numerous disadvantages. A major disadvantage for example is the need for a licensed copy of the modelling software with the same version as the native model project file, on each computer on which the Unity project is deployed. Unity has some limited support for a few simple CAD formats like DXF and SKP. Regarding advanced communication interfaces like WebSockets, OPC UA or ROS, Unity does not have native support. There are however plugins to add those interface capabilities. Unity does support basic VR capabilities like HMDs, but it lacks clustering capabilities and support for advanced tracking systems. Third party products like MiddleVR do provide full support of advanced VR hardware systems with Unity.
All in all, Unity is a very powerful and intuitive authoring tool for virtual environments, but it does have a few minor design choices that may have some disadvantages. For example, the play mode necessitates to restart the game for each test, which may result in a slower development process as opposed to editing during run time. Another example is the choice of scripting languages, C# is currently hyped, but it is closely linked to the Microsoft world. This makes it more difficult to reuse code elsewhere or recombine other code bases into the Unity project. Regarding its VR capabilities, Unity has to be used with MiddleVR to deploy on distributed visualization systems. Using MiddleVR to deploy a Unity project in a cluster environment has some limitations. First, the Unity project has to be deployed on every node of the cluster. The synchronization is based on making sure the behaviour of the Unity application is deterministic. To achieve this, MiddleVR synchronizes user input, random seed values and network communication. This means that the user has to manage the synchronization, especially for physicalized bodies, and handle the output of simulations and external systems consistently. Another major disadvantage of Unity is that it is focused on game design and lacks many features for engineering applications, especially interfaces for data import or to communication systems. There are plugins that are used to alleviate this, but they are of variable quality and performance. The user has to manage the extensions he uses, take care of versions, deployment and interoperability.

Unreal Engine

The Unreal engine [Unr] by Epic Games is one of the most popular and widely used game engines for professional game development. Most of the games using the Unreal engine are first or third person action games, with a major focus on combat mechanics. Those high-end production games are the main target domain of the engine. The engine is very flexible, making it easy to implement unique game designs. For a game development beginner there are easier engines to start with.
The Unreal engine does come with a graphical programming editor, but it is quite limiting. The better way to implement an application using the Unreal engine is by using C++. This is very low-level development, there are no in between scripting environments. The development workflow is again, similar to Unity and most game engines, based on an edit and run mode. Additionally, there is a 3d scene editor viewport. The scene editing, especially the composition of the scene with assets, is done in the editor, and the application logic, the behaviour of the active scene components, is programmed in C++ in Visual Studio. This results in a strange back and forth workflow between those tools. The rendering capabilities of the Unreal engine are stunning. It is known for its high realism graphics. The rendering pipeline uses deferred shading, global illumination, lit translucency, and post processing as well as GPU particle simulation utilizing vector fields. The physics engine used by Unreal is the PhysX engine from NVIDIA. There are no additional simulations for kinematic systems or mechanics. There are also no native reasoning systems, but this can be added as third party C++ dependency. It is surprising that the only geometry import formats of the Unreal engine are OBJ and FBX files. No CAD formats are supported. The support for advanced communication interfaces is not provided. But as for additional simulation modules, it possible to add any interface as third parts C++ library. The virtual reality capabilities are quite meager. Unreal does only support mainstream head-mounted displays.
To sum it up, the Unreal engine is one of the best gaming engines, especially regarding graphics quality and performance. As for Unity3D, it does lack CAD data format interfaces, advanced communication interfaces like WebSockets, ROS or OPC UA, and support for advanced VR technologies. There are much less plugins available for Unreal than for Unity, but this is negligible because the Unreal engine relies on low-level C++ implementation. The downside is the expertise needed to develop in C++, slower development overall and the development workflow is more complicated. But the huge advantage is the possibility to integrate any third party library. Overall, the Unreal engine lacks the focus and features for engineering applications and requires too much low-level development know-how to be attractive for engineers.

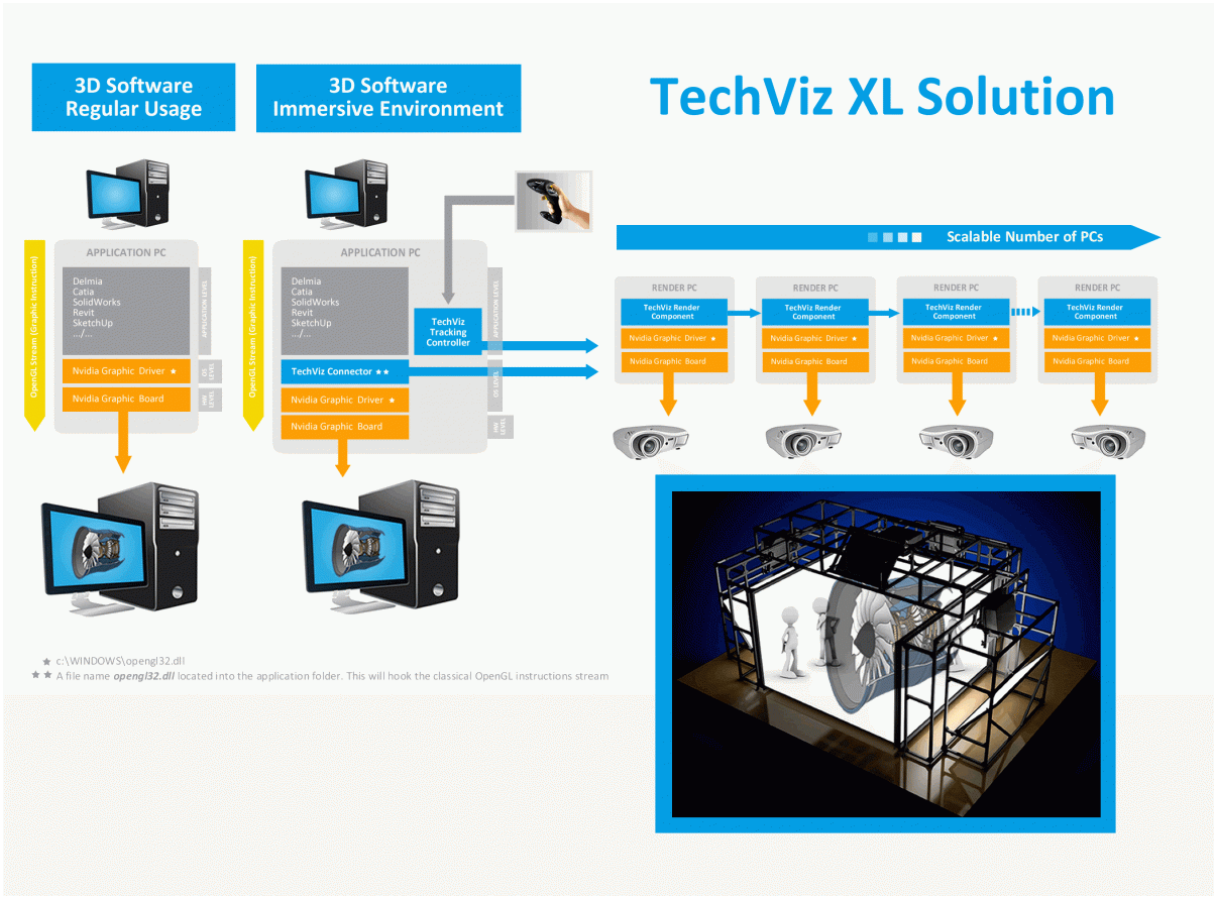


Fig. 3.1: TechViz solution, source: [Tec]

3.1.2 Virtual Reality Engines

In contrast to gaming engines, dedicated virtual reality engines have a focus on engineering applications. They do not have a broad user base and often are exclusively used in academia and industrial contexts.

TechViz

TechViz [Tec] is a virtual prototyping software, mainly targeting the product development process in mechanical construction industries. It allows to distribute the native visualization of 3D modelling and CAD software tools to immersive virtual reality hardware setups as shown on fig. 3.1. The approach taken to achieve this is quite special. By adding a backdoor in the custom compiled OpenGL driver library, it is possible for TechViz to switch the system library with their modified version which allows them to intercept the data stream coming from the application on its way to the graphics card. This allows TechViz to transmit the data over a network to rendering slaves, thus deploying the application data in advanced immersive hardware systems like CAVEs. TechViz also handles tracking data and input from VR devices to explore and interact with the scene data.

This method has interesting advantages compared to the classical approach of scene graph or input based synchronization. Firstly, it allows to use the native CAD software and avoids data transfer, data loss or additional workflows. An example is presented in [BVHO15]. Second, as it does not necessitate know-how to deploy data in the immersive system, making it very attractive for SMEs.
There are however also major limitations of this method. The first issue is that the data transfer is unidirectional. This means that there is no communication from the immersive VR system to the host CAD application. This limits the possible features of the VR environment to the generic features of exploration and primitive interaction like drag and drop or clipping planes. No higher application logic can be implemented.
Overall, the technology implemented by TechViz offers a great extension of CAD software to deploy models in immersive environments. However, there is nearly no possibility to extend the system any further than its generic exploration and manipulation features.

COVISE

COVISE [Cov] is a distributed software environment with simulation, post-processing and visualization functionalities. COVISE is strongly focused on scientific visualization. COVISE was designed to allow engineers and scientists collaborative working, spread over a network infrastructure. An application is divided into several processing steps, which are represented by COVISE modules. These modules, running as separate processes, can be spread across any network system, especially high performance infrastructures such as parallel and vector computers.
COVISE has different approaches to building virtual environments. The first one is to import complete virtual environments with interaction logic via VRML files. The second method to create application logic is using the visual programming module. The last method is using a Python scripting module. There is no editor provided, an external editor has to be used. The finished script has to be run using COVISE in a console window. For more advanced uses, additional modules have to be programmed.
The VR rendering modules of COVISE is named OpenCOVER. It is based on the OpenSceneGraph library. It offers only classic OpenGL forward rendering.
OpenCOVER supports 3D polygon mesh based file formats like 3DS, OBJ and VRML. OpenCOVER does also support a CAD based file format, DXF. When provided with the input of other modules, the OpenCOVER module can display geometric data from those modules, typically the visualization of simulation results.
COVISE has modules for rendering virtual environments on immersive hardware systems like workbenches, Powerwalls or CAVEs.

Overall, COVISE is very focused on scientific visualization. It is a strongly modular and decentralized system that allows to distribute the workload on many network resources. It is thus very complex to use and is not very suitable for creating classical virtual environments beyond the scope of scientific visualization.

IC.IDO

IC.IDO [ICI] from the company ESI Group offers a typical virtual engineering solution. It is strongly focused on providing virtual reality technologies to the machinery engineering sector, especially for the product development process. It provides the infrastructure and tools for the design review of construction data in an immersive collaborative environment.
IC.IDO does support high-end visualization systems, including distributed cluster environments and advanced tracking. It also supports state of the art HMDs.
Overall, IC.IDO is a software package for virtual engineering applications, thus closely related to one of the use-cases in this thesis. It is a high-end professional software, combined with the necessary consulting and customizing needed for introducing virtual reality technologies in industrial ecosystems, especially for the product development process. The major downside is that IC.IDO is not a generic authoring system for virtual environments, they do not provide an engine or framework, but specialized applications for virtual engineering. It is also difficult to obtain more detailed technical information as IC.IDO is completely closed source and has no big community like Unity or other gaming engines.

3.2 Virtual Reality Hardware Systems

This section focuses on the immersive hardware systems and the human machine interaction devices that are typical for virtual reality applications.

3.2.1 Tracking Systems

A tracking system is responsible for detecting and tracking the motion of objects or people in space. Usually the focus lies on tracking the user, his head and hands, as well as his surroundings. The systems differ in what and how they detect. Some are based on external sensors like cameras and cover a specific area. Others are attached to the user and can track in his vicinity while he moves around. Tracked objects can be rigid bodies like devices used to interact with the virtual environment. Those devices usually have additional input capabilities like buttons and joysticks. To track the user head position it is often necessary to wear glasses, usually also necessary for stereoscopic displays. In addition to rigid bodies, there are systems that allow to detect objects without a fixed geometry like the human hand or the posture of a person. And then one can differentiate between recognizing hand gestures for inputting discrete commands, and

(a) (b)

Fig. 3.2: a) ART camera, b) Flystick interaction device, source: [ART]

precise detection and tracking of the configuration of the hand with its fingers. In this section some tracking systems and input devices will be presented.

ART System

The ART system [ART] (fig. 3.2) uses cameras to track objects. The cameras are surrounded by infrared LEDs that send out flashes into the tracking space. The IR light is reflected by markers attached to objects. Those markers are small spheres with a highly reflective surface. This allows the cameras to easily detect the markers. For each object there are multiple markers attached to track the object position, but also its rotation. The marker configuration does also allow to identify the objects. The primary interaction device is the Flystick, a tracked object with a trigger, a few buttons and a joystick. The benefit of the system is that it is very robust and quite precise. It is overall well establish in high-end visualisation systems like CAVEs or Powerwalls. The only major drawback is the high investment costs.

HMD tracking systems

Most HMD (fig. 3.3) tracking systems have nowadays a systems that tracks the head and hands of the user. The HTC Vive [HTC] uses external beacons to localize the tracked objects using multiple photo receptors on each object. The tracking is very precise as it uses external reference points. The Microsoft HoloLens [Mica] on the other hand uses the information of its depth camera to compute its motion. It uses SLAM to map the surroundings of the user and position the camera simultaneously, thus tracking the users head. This method is very computationally intensive and does not easily allow to track other objects like hand held devices. The HoloLens does detect the user's hands and allow him to interact with his surroundings.

(a) (b)

Fig. 3.3: a) HTC Vive, source: [HTC], b) Microsoft Hololens, source: [Mica]

Microsoft Kinect

The Microsoft Kinect [Micb] is a depth camera that allows to detect persons and track their full body posture. It is though quite inaccurate and does not detect the hand configuration due to its limited sensor resolution.

Leap Motion

The leap motion [Mot] is a small device that uses a depth camera to capture the motion of hands. It is designed to be used by a single user and detect the configuration of his two hands. One major drawback of the leap is that the use of multiple leap devices is not supported.

Myo Armband

The Myo [Lab] is an armband with eight electrodes that get in direct contact of the user's arm and measure the muscle activity. Based on the muscle activity the device deduces the configuration of the hand. The system works well but it has a major drawback. The system can only detect muscle activity, the muscles have to be tensed up. This imposes a strain on the user, making the device not usable in a continuous manner.

Summary of the Tracking Systems

The most commonly used tracking systems nowadays are based on optical sensors. Older systems did use mechanical, acoustic or electromagnetic sensors, but they have been shown to have too many disadvantages compared to optical systems. The most relevant system when considering the use in CAVE systems in an industrial context is the ART system as it is very robust, reliable and accurate. It also offers the necessary interaction devices. For HMDs similar solutions are available from the HMD producers. Older HMDs did use inertial sensors which were inaccurate and over time had a drift of the reference coordinate system. Regarding the various

input devices, they all have their limitations that make them difficult to employ for generic use-cases. The ART Flystick is here again the best choice for a generic interaction in VR.

3.2.2 Distributed Visualization Systems

A distributed visualization system is the combination of multiple display devices to form a more complex compound display system. The devices are usually the same type, often placed edge to edge to form a big field of view, but technically there are many possible configurations with different display types, arranged in space as needed.
It is often necessary to use more than one computer to run such a complex setup, a PC cluster can be used to support the graphic cards necessary to feed all the displays. The master/slave concept is typical terminology in this domain to describe the relationship between the master application that often runs on a dedicated PC, and the salve applications that only produce the visual output to be displayed.
To be able to be deployed in a cluster environment, an application has to support clustering. Clustering is rarely supported by 3D engines as it drastically increases the complexity of the application, and most engines are not targeted to distributed systems. It also requires the deployment on all cluster nodes and the configuration of the network, usually on the master application. The display system is configured by specifying the position and dimension of each display. The coordinate system chosen is usually the tracking coordinate system. The combination of the display parameters and the position of the user allow to render a virtual scene from the perspective of the user, this is referred to as head tracking.

3.3 Open World Generation

Open worlds are iconic virtual environments. Their size and quality grows with every hardware and software advancement to demonstrate the performance of new graphic cards or new graphics engines and rendering algorithms. One can differentiate between modelled open worlds and generated open worlds. Millions of dollars of game development budgets can go into designing beautiful open world environments. Generated environments on the other hand can be realized with much lower budgets because, once the algorithms and assets are created, they are reused and parameterized to create open worlds of any size without additional costs. There are other advantages to generating virtual environments, for example will they contain much more semantic information than modelled environments. This section aims at analysing the state of the art of open world generation systems, especially for the road network.

3.3.1 Cities Skylines

Cities Skylines is an open-ended city-building simulation. The user plans and controls the urbanization, defining construction zones, constructing the road network, managing taxation as well

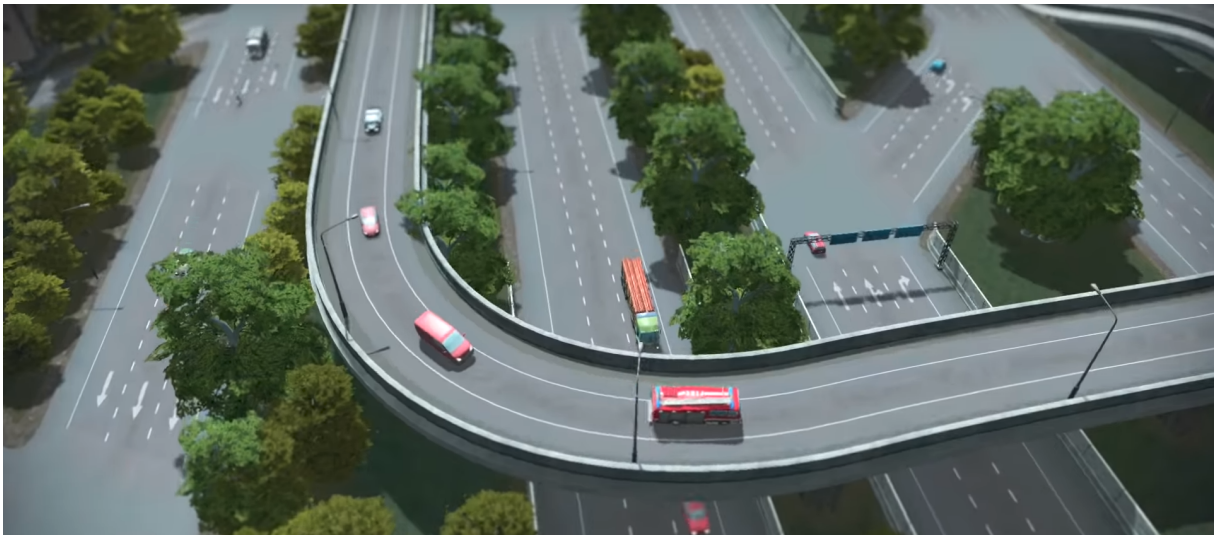


Fig. 3.4: Cities Skylines road network, no markings at the road intersections, source: [Int]

as public services and transportation. Further constraints are the city budget, pollution levels as well as population indicators like health and employment. The game engine is designed to be able to simulate the daily routines of nearly a million unique citizens. The simulation allows for complex dynamics like realistic traffic congestion, and the effects of congestion on city services and districts. The most advanced feature is the graph-based road network. This makes this game one of the most realistic generated open world environments available. Most other applications use modelled worlds or greatly simplify the spectrum of variations of the road and intersection geometries.

The road network and other urban elements are created by the user. The user edits the road graph, points and edges, the system then generates the road geometry and texture based on the graph and meta information. The geometry is basically a sweep model. From the visualisation, it appears that the road lines are short textures spanned along the road geometry using UV coordinates based on road length. This is efficient, but does not work for road crossings as seen on figure 3.4. Road crossings have no road markings as this would be highly inefficient because the lines depend on the angles of the roads entering the intersection, which can be arbitrary. This issue is fundamental for real-time open world applications that use a generic graph for the road network. For a city building application it might not be very important, but for a driving simulator it is crucial.

Even after extensive research, there does not seem to be any application on the market that goes beyond Cities Skylines. The projects found in academia ignore this problem completely.

3.4 Virtualization

This section aims at presenting the state of the art virtualization workflows, especially the CAD software and the data interfaces. Furthermore, projects that employ reasoning for data integration and virtual environments are analysed.

3.4.1 Computer Aided Design Software

There are many state of the art CAD systems, the system employed by a company usually depends on its size and sector. The focus in this section is set on CAD software used in SMEs in the plant engineering sector. The aim is to analyse the possibilities to extract the CAD model data from the modelling tools for further use. There are two ways to obtain that data, on one hand it is possible to use generic export formats, on the other hand one can use the plugin systems of the CAD tool that allows to access directly the entire native construction data model. Those aspects are presented below for various CAD software, structured based on the domain the CAD tool is mainly used for.

Mechanical CAD Software

The most important mechanical CAD systems for the plant engineering sector are SolidWorks, Siemens NX and PTC Creo. SolidWorks is most popular in SMEs.
The first aspect to discuss are the export formats. All three tools offer very similar variety of export options. The important features to compare those formats are if they are CAD or mesh formats, if they store assemblies or only parts, and how well are the CAD models preserved. The mesh formats are VRML, OBJ and STL, all available with the three tools. The CAD formats are STEP and IGES, also all available with the three tools. Siemens NX and PTC Creo also offer the JT and Parasolid formats. The way the CAD tools store information in those formats is nearly identical, this makes it possible to just compare the file formats without going further into specifics of each CAD tool.
As shown in table 3.1, none of those exchange formats support advanced features like kinematics or features. Not every format is able to export assemblies and even less data sets do contain the product structure, not because the format would not allow it, but because the CAD tool does not write the information into the exported file. To obtain access to this information, it is necessary to create a plugin to gain access to the native data model of the CAD system.
SolidWorks does offer an extensive plugin system using either VB, C# or C++. The system allows to access the whole data model, including the product structure, geometries, features and materials. Siemens NX does provide an interface called NX Open to write plugins in many languages, C++, Visual Basic, C#, Java, and Python. Creo offers free APIs for the development of plugins in VB, JavaScript and Java, but they are limited in functionality compared to the paid toolkits for C++ and Java development.

Format	Type	Assembly	Structure	Kinematics
VRML	Mesh	yes	no	no
OBJ	Mesh	yes	no	no
STL	Mesh	no	no	no
STEP	CAD	yes	yes	no
IGES	CAD	yes	yes	no
JT	CAD	yes	yes	no
Parasolid	CAD	no	no	no

Tab. 3.1: CAD exchange formats, do they export assemblies? Do they conserve the product structure and meta data?

Building CAD Software

Building architecture and construction design is of similar nature as mechanical CAD as the goal is to describe the geometric representation of a product, int this case a building. There are still major differences and a whole other set of tools because building and MCAD domains have a different history and building CAD did first evolve towards 2D construction systems and is only slowly moving forward to 2.5D and 3D systems. Modern building CAD software is integrating more semantic and meta information in CAD drawings, the so called building information model (BIM). A good example is the Revit software from Autodesk. The main data model and exchange format for BIM data are the industry foundation classes (IFC). This data model is intended to describe building and construction industry data. The IFC file format is platform neutral and its specifications are openly available as official international standard ISO 16739:2013. It is commonly used as collaboration format in architecture and construction engineering.

Electric CAD Software

ECAD systems describe the wiring of a machine and its electronic components. The wiring contains different kinds of cable connections that form graph structures. The most prominent one are the electrical circuits, typically accompanied by a bus system like Profinet. The important aspect for the virtualization is how to access that data, map the eletrical components to MCAD components and implement the wiring logic. The ECAD data model is different from MCAD and BCAD data as it usually does not represent geometric data but primarily topology data. The most important electrical CAD system for the plant engineering sector is EPLAN which has a quasi monopoly. EPLAN offers various export file formats, XML based export files that contain the serialized data of the wiring graphs. The data is stored in human readable form and

can easily be interpreted. It is also easy to read the data and automatically analyse it for further building the virtualization.

3.4.2 CAD to VR

The basis for virtualization is to extract the MCAD, ECAD and programming data from the corresponding CAD tools and fuse it together in a single data model. The first step, extracting and gathering the data, is very difficult as CAD tools offer very limited export functionality as described above. Combining the MCAD and ECAD data as for mechatronic systems is challenging because the modelling is done in different software tools, by different engineers, which does not encourage using unique identifier across domains. Various works are trying to develop methods to address this issue [AB07, AB08, Bel19].

Kinematics

As described by Lorenz et al. [LRP+15, LSR+16], most formats do export the hull model, but lack the dynamic information like constraints and kinematics. Lorenz et al. do propose the development of plugins to solve this issue, but this has its limitations as it requires a plugin for every CAD system, thus a lot of development and maintenance. Another approach is to analyse the CAD model components and deduce the kinematic chains.
This has to be done in two steps. First, segment kinematic components using geometric analyses, then assemble the components into a kinematic graph structure based on topological analyses of the constellation of the components. Current works show basic feasibility of such an approach. Lupinetti [LGMP16] et al. describe a method to identify the dynamic relations between parts. Further in [LGMP17], a method is described to identify rolling bearings in assembly models as a first step towards the automatic functional characterization of CAD components in assemblies. Strahilov et al. [SOB12, SO+12] present a virtualization of an automated assembly system using a physics-based approach.

Electrical CAD

MCAD does provide the shape and placement of machine components. This can be for example armatures, fixations or kinematic elements. But also mechatronic components like switches, buttons, interactive panels, and all the various kinds of actuators are represented in the MCAD data. Those components are powered by electricity and connected to integrated control units, but this information is not modelled in the MCAD data. The electrical CAD model contains exactly this missing information, the wiring of the components. The issue is that the MCAD and ECAD models are not meant to come together. Once developed, the plans are printed on paper and used to construct physically the whole machine. The parts are machined and assembled, then the wiring is added, only by workers reading and interpreting the plans. This means that

there is no straightforward approach to integrate MCAD and ECAD data into a single consistent model. Pollari [Pol15] addresses this issue in the scope of a mechatronic product with a circuit card assembly and casing model. The suggested approach integrates the ECAD and MCAD data in a STEP AP-239 standard. Then only a consistency check is performed on the integrated data model, and not a full virtualization. Emmer et al. [EFJS15] acknowledge the importance of the ECAD/MCAD collaboration in the product development process and discuss the evolution of CAD tools to provide the necessary features for integrating both worlds. Prof. Ovtcharova [OMG+11] presents a function oriented, ontology based approach to provide a structure that models the functional interdependence between mechatronic components for semi-automatic reuse of product functions and mechatronic components. There are also benefits of virtual environments for embedded software [HO13], for example to validate the software development for mechatronic systems.

Building Information Model

Building CAD model data is very important for the virtualization process. Machinery, production lines or logistics need to access building resources like electricity, or even just make sure they fit geometrically. Wicaksono et al. [WSR+11, WRO12b, WRO12a, WO12, HHWO13, WBO13, WDKHR13, WJRO14] use semantic web technologies, ontologies and reasoning, to analyse in the context of BIM, energy efficiency in building management and manufacturing.

Application Logic

Solving the issues of the steps above would result in a virtual model with geometries, kinematics, and the wiring. The last missing aspect for a complete functional model is the application logic. The goal is, as for the MCAD and ECAD integration, to fully automate the addition of the application logic. Automating the application logic is the most complex part of the virtualization process. First, one needs a way to define and store generic knowledge about the application domain. Then the application data model, in this case the CAD model, has to be semantically enriched to map the generic knowledge onto each component in the scene. Furthermore one needs a system that can apply the domain knowledge on the application model in real time. Semantic web technologies, more specifically ontologies and reasoner, are designed to achieve this, but are rarely used for real time applications or even virtual reality applications.
There are some works that try to use reasoning to automate the generation of application logic for specific use-cases. Siddique et al. [SR97] use automated reasoning techniques to generate a disassembly process model for a virtual prototype, but require interactive input to complete the disassembly evaluation. The work of Chang et al. [CCKS05] successfully uses ontologies to model an explicit representation of a virtual environment, enabling character agents to reason in the environment by inference. They go as far as to show that such a character can have basic

planning capabilities without any implicit knowledge of the scenario, resulting in a better adaptability and creative use of the environment. Grimaldo et al. [GBL06] as well as Messaoud et al. [MCSG15] use ontologies to provide intelligent virtual environments with semantic information and define general and reusable activities for the simulation of virtual agents. Schneider et al. [SWO19] propose a method with a modular domain ontology that formally describes cyber-physical systems. Ovtcharova et al. [OMK06] describe a software application for capturing and re-using rule based knowledge concerning manufacturing machine services like advanced task and process planning, machine configuration, maintenance, training and management support. Furthermore they support different kinds of manufacturing machines and manufacturing machine specific domains. Kiesel et al. [KKB+17] propose an extension of the AutomationML with support for ontologies. They aim at instantiating virtual commissioning models using a rule-based method.

3.5 Summary of the State of the Art

3.5.1 Authoring Software for Virtual Environments

To develop virtual environments, there are clearly two families of software systems, gaming engines and virtual engineering engines. Both worlds present their advantages and disadvantages. Gaming engines offer powerful toolkits for the creation of virtual environments, high-end rendering and polished authoring environments with low entry barriers as well as big communities, documentation and support. The downside of gaming engines when used for engineering applications is the lack of advanced communication and data exchange interfaces as well as advanced simulations and system interfaces.
Virtual engineering software does support high-end virtual reality technologies and depending on the system provides advanced communication and data exchange interfaces. On the other hand they lack flexibility as they are often specialized in a very narrow domain. They also lack the state of the art rendering and application authoring systems that gaming engines can provide. Overall, one aim of this thesis is to create a solution that recombines the most important aspects of each world, creating a system that allows engineers to focus on the virtual environments they need, while abstracting the low-level software development aspects and providing the interfaces to setup complex data exchange workflows.

3.5.2 Open World Generation

For many use cases it is imperative to automatically generate semantically rich open world environments with realistic road networks. For example for driving simulators or testing environments for self-driving cars. To achieve a realistic and varied environment, it is important to use a generic planar graph structure as basis for generating the road network. Such a graph can

be modelled by hand, or imported from available map databases like OpenStreetMap. A major challenge is then to generate the visual representation of the intersections as seen in section 3.3.1. This issue will be addressed in this thesis, allowing to draw any road markings on any asphalt surface anywhere in the open world environment and still keep the real-time performance.

3.5.3 Virtualization

The virtual twin [Ovt15] is the holy grail of virtual engineering, but the main bottle neck is the underlying virtualization workflow. The first aspect of the CAD virtualization workflow is to analyze CAD software in regard to export formats and plugin interfaces to get access to the construction data. Then comes the integration of MCAD, ECAD and the automation of application logic creation. CAD systems allow to export geometric information in many formats, but only the STEP format seems to be a reasonable option for the export of geometry, product structure and meta data. For dynamic and kinematic information, there is no export format available that contain this kind of data. The STEP specifications do support it, but the CAD tools do not add the information to the exported data. To access the full native CAD data, one has to implement a plugin. This has to be done for every CAD software separately, which greatly increases implementation and maintenance costs. Regarding the integration of MCAD with ECAD data, there are some works that express the need for solutions to that problem, and some that try to achieve this for collaboration uses, but none goes as far as to build a functional virtual model. The same is true for the automated integration of application logic based on semantic web technologies. A few works do address the topic in the context of product development, others address the topic of introducing reasoning in virtual environments for specific uses, but none goes as far as using reasoning to create a fully functional virtual model.

4 Methodology - Engineering Virtual Reality

Virtual reality is a broad and thus very challenging research field where interdisciplinary expertise is key. The difficulties encountered when creating virtual environments for engineering applications have been motivated in the introduction chapter of this work. The state of the art design process is too slow and requires too much know-how in computer graphics and software development. Research problems and questions have been defined that will be discussed throughout this thesis. This chapter tries to address the research questions by presenting a methodology to support the creation and usage of virtual environments, especially in the engineering context.
The research questions are:

- How to increase efficiency of virtual environment authoring for engineering applications?
- How to reduce the amount of work for open world virtual environments?
- How to automate the virtualization process for virtual engineering applications?

To answer the first research question, we propose a VR system design focused on a seamless authoring process to make the creation of virtual worlds more efficient and intuitive. The proposed system design has to be fully modular and highly extensible. A core component is the integrated development editing functionality. It allows to easily combine internal and external modules to maximise the synergistic use. Various modules, tools, simulations and interfaces are presented that are often required for various engineering applications. We also propose to use semantic web technologies to further ease the development of rich interactive virtual environments. The focus of the authoring process concept is of course the development of real-time interactive applications and the deployment in distributed visualization environment with tracking systems. All modules and features have to be designed with this requirement in mind.
The second research question addresses the high costs of content creation for immersive virtual open worlds. We propose a system that uses advanced algorithms to create geometric assets based on real world GIS data. Topography and map data is freely available as open data. Using this data as seed for world generation algorithms allows to create organic worlds with realistic road networks and building layouts. This also comes with many challenges as map data contains nodes with an arbitrary layout of incoming and outgoing edges, resulting in numerous special cases for road intersections.

The last research question revolves around the virtualization process of CAD data. Integrating mechanical CAD, electrical CAD and automation is necessary for creating fully functional virtual models. Even when done manually, the virtualization is very costly and greatly limited in its dynamic and functional features, it often lacks intelligent behaviour. We propose to use semantic web technologies to automate the creation of functional mock-ups and prototypes. This technology can also be used to automate the parameterization of functional and simulation modules needed for different features of the virtual model like multi body systems or the simulation of the electric wiring. A core components of the proposed concept is the combination of the knowledge base with the scene-graph and the reasoning system. A complementary aspect is the deployment of virtual engineering methods in SMEs. We propose a basic approach to integrate VR technologies seamlessly in engineering processes and enable companies to use VR productively without the need for dedicated personnel. The aim is to build on the automation of the virtualization process to further reduce the duration of engineering and validation cycles in the product development process. This is important to push the front loading boundaries and raise acceptance of such technologies.

4.1 Virtual Reality Authoring System Design

Authoring virtual reality applications is a tedious task, adding 3D assets, dynamics, designing lights and shadows, but even more important, programming the application logic. This includes navigation and interaction paradigms as well as animations, events, simulations and intelligent behaviour. The know-how needed and the necessary time investment is very high, making development very costly. Finding a way to automate, even partly, this complex creative process would open up virtual reality to many new fields, especially for engineering applications in SMEs.

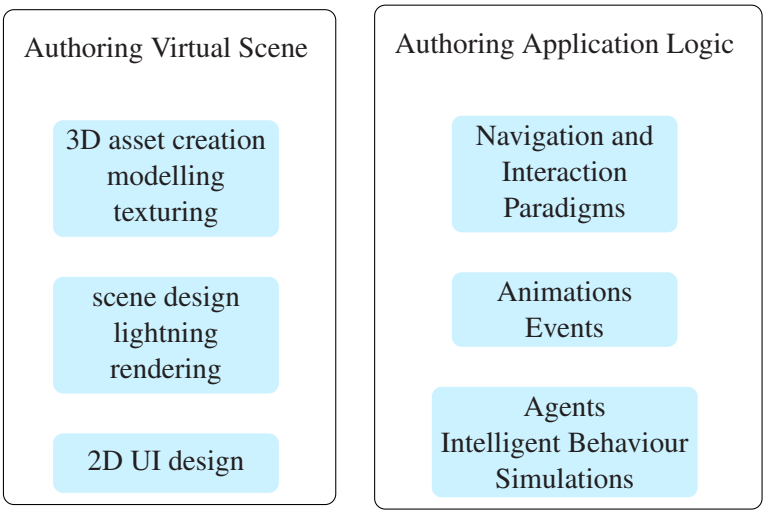


Fig. 4.1: Authoring Virtual Environments

The first step is to analyze the authoring process as depicted in figure 4.1. As the focus of this work lies on engineering applications, the asset creation will not be addressed in depth. In most cases it will be CAD data or scientific simulation data, but may of course be any 3D models, imported via one of many exchange formats. The other aspects will be analyzed in detail in the following, while focusing on the CAD virtualization workflow, especially for huge machinery like industrial production plants.

4.1.1 System Implementation Concept

When starting a software project, it is important to decide on a strategy to support the many decisions one has to make. The strategic basis is to define the intended application area and use as well as the target audience and hardware systems. The application area is obviously virtual reality for engineering applications, this requires a complex software system with low-level hardware interfaces, especially to the graphical processor, but also high level communication interfaces, virtualization and validation modules and much more. The target audience are engineers with little or no coding experience. This can be a student who participates in a practical course or uses virtual reality for his thesis work, or it can be a construction engineer in a company, using the system to virtualize the current product development progress.
It is also important to consider legal issues relating to copyrights and licenses. A potential new code dependency has to be carefully evaluated, as it may result in legal issues if the license does not harmonize with the license of the main system. A third party dependency can also be discontinued or even deprecated. This can mean a lack of support and further development. The focus was set on a long term strategy, emphasizing sustainable development, resulting in a very modular system architecture. The system adds different abstraction layers to greatly increase the reusability and synergistic use of code, features and modules.

Open Source Development

An important question is what legal frame to choose for a new software package. In the case of PolyVR, it was clear that the choice would be to go open source, the importance of open source is shown on fig. **??**. There are many reasons for this choice, some purely idealistic, but also many practical and even business oriented. The more abstract and idealistic reasons are mainly to make it accessible to everyone, to add a piece to the world software heritage. This heritage is the combined worldwide effort of many engineers over decades, the basis used in research but also under the hood of so many industrial software systems, even closed source. The current advancements in technology would not be possible without the solid foundation the open source developments provide. A myriad of libraries provide a stable and well tested code basis, enabling developers to handle the ever growing complexity of applications and develop cutting edge technologies.

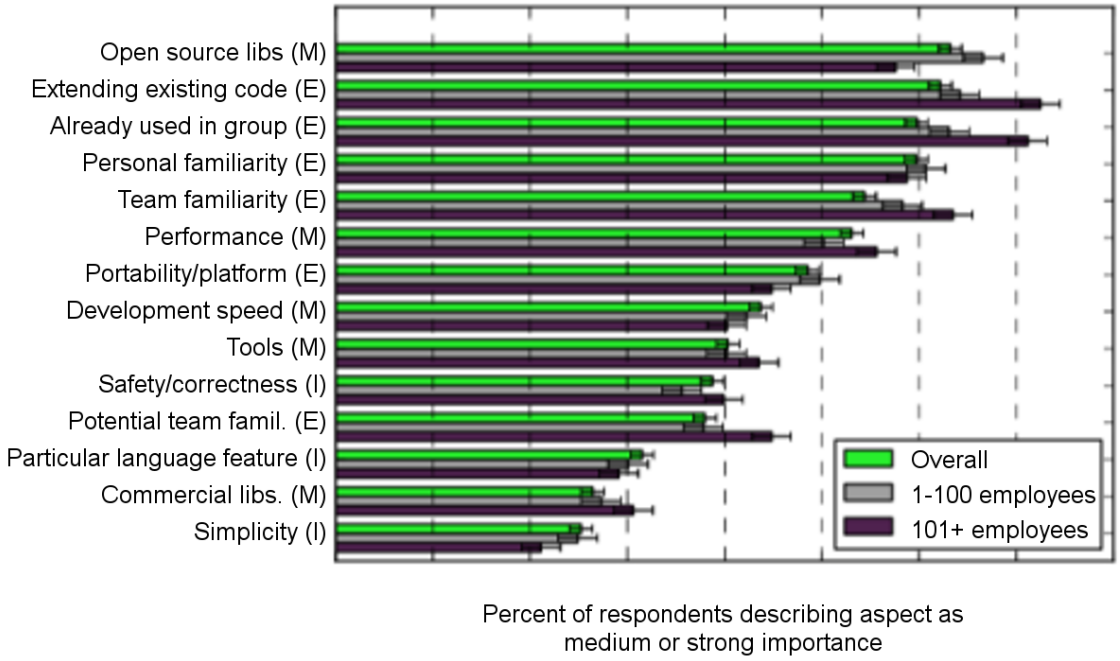


Fig. 4.2: Survey on important aspects of programming languages show the importance of open source, source: [MR13]

The more practical reasons are for example the free access to a huge infrastructure available for open source projects. This makes development and third party contributions much more efficient, but also deployment and maintenance. The transparency also reduces the general amount of bureaucracy and greatly simplifies many workflows. Another very important reason is the compatibility with more restrictive open source licences. Being open source allows to use software with those restrictive licences, it gives essentially access to the full range of open source libraries.

But in the end it is important to address the concerns of the customers. One has to keep in mind that even when working for the industry, it is possible to use open source and to keep everything project related, the developments, company data, and the overall IP of the company closed and secure. Even closed source features can be incorporated in a specific solution to a certain degree. And even for a company there are many benefits to using an open-source solution. They directly benefit from past developments and can easily ask for customization. They also have access to the source, which is somewhat a guarantee to a sustainable investment.

Software Architecture Concept

The concept for the system architecture is depicted on 4.3. The core module of a virtual reality software is the scene graph. There are various scene graph libraries and 3D engines available, most focus on desktop environment and especially gaming applications. In addition to the

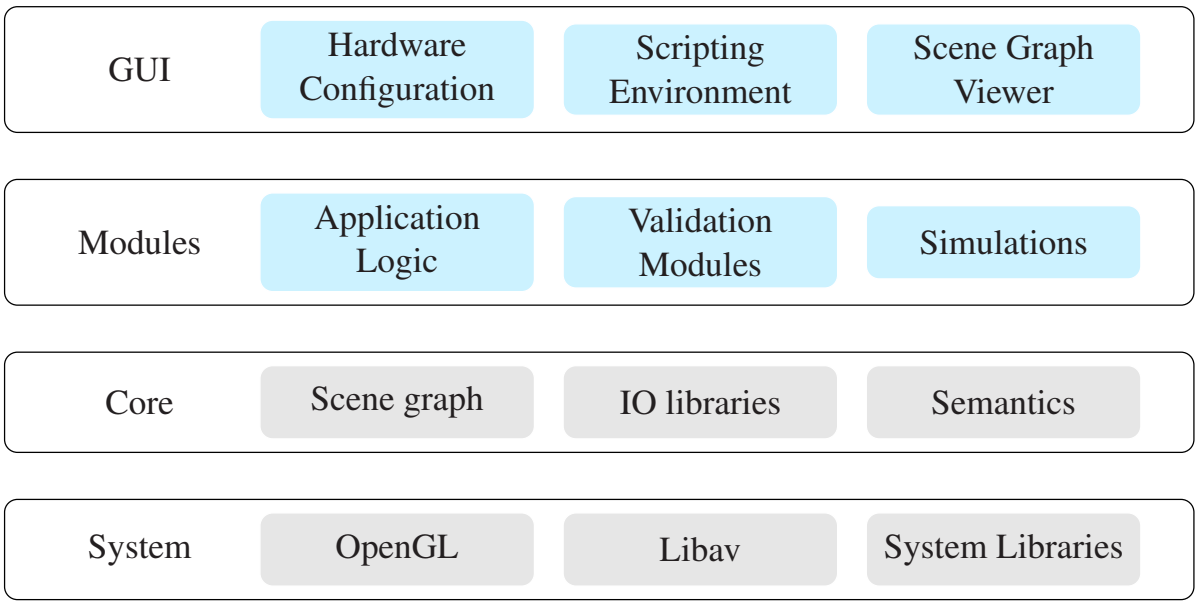


Fig. 4.3: Software architecture concept for PolyVR

features found in all scene graph libraries, the scene graph employed in this work will need to support distributed visualization systems and state of the art tracking systems to allow the deployment of applications on CAVE systems. The overarching structure of PolyVR are the abstraction layers that greatly simplify the use of the system, but also eases the modular development. The highest abstraction level is the front-end, the GUI with integrated scripting environment that allows to easily develop virtual reality applications. The second layer is filled with various modules, providing all the necessary features for engineering applications. Those modules use the core scene graph module, situated in the third abstraction layer, right above the low-level hardware libraries like OpenGL and Libav.

Programming Languages

An important choice to make is about the programming and scripting languages that should be used for the virtual reality system, but also available to the users to create the virtual environments. Important factors for the adoption of a programming language are for example existing code, the familiarity with the language and its widespread use overall as seen in fig. **??**. Furthermore as depicted on **??** the trending language for core development is C++, Python as a widespread and popular multi-purpose scripting language and JavaScript for web development. This set of languages is an important strategic choice in this thesis. Those are the languages that this work is relying on to build a sustainable ecosystem of inter weaved technologies. C++ is used for the core development of the VR framework, Python is the scripting language offered to the user to author the application logic of virtual environments, and JavaScript, CSS and HTML are the languages to design 2D UI elements for the virtual environment.

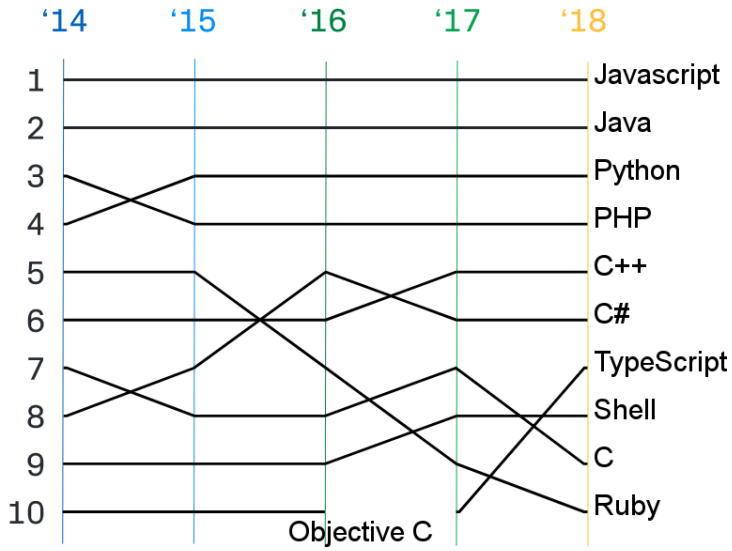


Fig. 4.4: Language statistics from GitHub, source: [Git]

Further reasons to choose Python as a scripting language are that on one hand Python is a very intuitive and flexible language, but on the other hand that it went through a hype resulting in a broad spectrum of available libraries with python bindings. It allows to easily interface to the core functionality of PolyVR written in C++, allowing the user to access all the core modules of PolyVR. Python is quite slow, but this does not matter as it is merely the glue holding together the application logic. It allows to easily instantiate and configure the various core modules, which themselves do the heavy lifting.

When creating 3D environments, it is necessary to have access to the shading pipeline, introducing custom shaders to achieve various goals and effects. This can be done at run time, meaning the user should be able to author the shaders and directly see the result of each change in the 3D environment. This will greatly reduces the development time of such shader code.

Web technologies are very popular, even for non proficient developers. Using web technologies to design 2D UI elements has many advantages. The users may already be familiar with web development, they have access to a huge community to draw knowledge from and they can reuse the skills they acquire or deepen while working on VR development for future web projects. Two dimensional widgets are important features in virtual worlds. They are useful for instance for displaying information, meta-data of virtual data models, or for adding control panels for production plants, or even just adding a classical flat menu, attached to the camera node. It is important to facilitate the design and deployment of such widgets. For this purpose, it is very promising to create websites and put them on any 3D surface in the virtual environment.

4.2 Open World Generation

An open world is an extensive explorable virtual space that often reproduces elements from the real world like some kind of terrain with assets, dynamic entities and a few rules like simple physics.

4.2.1 Terrain

Topography is important to make open world environments interesting, a flat plane that extends in all directions is very unrealistic and can quickly break immersion. There are also cases where topography is important for the application, for example when using virtual environments for planning urban infrastructure or analysing geological processes. There are many ways to create a terrain geometry for a virtual environment, from modelling by hand using a 3d modelling software, over world editor software often used in gaming, to 3D scanned areas using fly over Lidar technology for example. Visualizing a vast geometry like terrain is challenging in that when using a mesh one has to struggle with the trade off between the resolution and a high polygon count. This is where tessellation shader offer the needed flexibility to use a low resolution mesh and combine it with displacement maps, so called height maps, to dynamically subdivide the mesh to visualize a detailed topography in the vicinity of the user. This does also work well with topography data as it is usually exchanged using raster data formats. When using GIS data it is important to manage the data, for example to convert from geographic coordinates to Cartesian coordinates or to manage the resources like maps and height data. When considering a planet, its surface has to be composed of many chunks. To allow real-time rendering only the relevant chunks are visualized.
The design of the open world system starts with a planet module that manages its surface by splitting it into many sectors, each a terrain instance with its height data and map data. The terrain module uses tessellation shader to apply the height data as displacement map. When the virtual environment is situated at a specific point on earth, the planet module puts the corresponding chunk at the center of the OpenGL global coordinate system. This is important to avoid jitter due to float precision errors, especially when introducing transformation matrices on planetary scales. All further assets like roads, buildings and nature are addressed in the next sections.

4.2.2 Road Network

The road network is the most complicated asset of a realistic open world environment because it is not localized but spreads over the whole terrain. As for the topography, it is possible to model a road network using specialized editors, but on such large scales it is a lot of work and may not be easily made to look natural. A road network is mathematically a planar graph, where nodes

are connected by edges in any possible way. Many editors use some kind of discretization of the graph, only allowing certain angles a road can connect to a graph. This greatly simplifies the generation of the roads and intersections geometries and their texturing, but it also looks much less natural.

The method proposed in this work is based on the use of GIS data. Typical map data contains a simple planar graph for the road network and polygons for buildings and ground patches or zones. the first step is to generate a road network based on the graph and meta data attached to the nodes and edges of the graph. Each road has a direction and a number of forward and backward lanes, where backward lanes are the ones that go in the opposite direction of the road. Based on the planar graph, a new graph is created that does not contain the roads, but each lane as a separate edge. This is important when using the graph for simulations or other applications using lanes. Certain lanes allow in reality lane-switches, this is integrated in the graph structure as relations between edges. To make the roads more realistic the nodes are enriched with tangents used to compute Bézier splines instead of linear edges. This is especially important for the generation of the road surface geometry and the road markings.

The most complex part of the road network are the intersections. They are entirely defined by the roads and more precisely their lanes, that enter and exit the intersection node.

Once the road network graph containing all lanes on roads and intersections is computed, the geometry can be generated and the texturing of the road surface. The geometry is easily generated using sweep model generation for the roads and polygon triangulation for the road intersections. The texturing is much more complicated. The challenge is how to visualize all markings on roads and intersections in real-time. The problem is that each intersection has unique lines, making it necessary to create a different texture for each intersection and road, which is simply impossible to do in real-time, at least with a naive texturing approach. A solution to this problem is to develop a special shader that draws the markings based on the mathematical description of the markings. The idea is to compute for each fragment the distance to the markings on the road and decide if the fragment is on a line or not, resulting in an according coloration. A key aspect is to choose a suitable mathematical representation to describe arbitrary markings in 3D. This work proposes to use 2D Bézier splines for two reasons, they are capable to approximate good enough any curved paths, and they are described by equations with degree two. This is important because the goal is to compute the distance to those curves in the shader, which is limited in its capabilities. The equation for the distance to a quadratic Bézier curve is of degree three, which is computable using analytical methods. If the Bézier curve was a cubic curve, thus of degree 3, the equation for the distance would have been of degree 5! To solve such an equation requires numerical methods, not suitable for an implementation in a shader. To transmit the Bézier spline data to the shader, the parameters of the equation have to be encoded in a texture, where each line corresponds to a road. It is thus possible to write the data for hundreds of roads and intersections into a single texture, allowing to share the OpenGL material between roads,

thus reducing the complexity of the scene. The fragment shader reads the parameters of all road markings for the corresponding road or intersection, and computes the distance to the curves until it either detects that the fragment is on a marking or it is certain that the fragment is not. Further typical effects are used in the shader to add noise to the asphalt as well as Phong lightning, resulting in a quite realistic looking road. A benefit of this method is that the markings are perfectly smooth, there is no tessellation of the curves as they are drawn using the mathematical description instead of an approximation by linear segments.
Other typical assets strongly linked to the road network are guard rails, kerbs or fences. Both are usually linear features found along the road, following its path. In map data they are represented as poly-lines. The geometry is, similar to the road surface, easy to generate using sweep models. The texturing can be realised using the linear position to define UV coordinates that wrap along the U direction, resulting in a simple repeating pattern.

4.2.3 Buildings and Street Signs

Buildings are often designed, either as entire 3D models, or in chunks or modules that can be reconfigured to add some variety to the virtual environment. The method used in this work is, as previously for the road network, based on map data. Each building is described by a polygon that represents its outline. This information and the attached meta data are used to generate the building geometry. The challenge of creating buildings is to make the generation algorithms robust enough to handle any shape of the outline polygon. In addition it is again very important to create efficient models, making varied and detailed models difficult to realise.
The method developed in this thesis is to use mega-texturing and multiple layers of texture coordinates. This allows to create a low-poly geometry for a whole district from the building outlines. Even thou the district is a single geometry with a single material, it is possible to create different types of buildings with various facades. This is achieved by using specific sets of UV coordinates depending on the type of building to be represented. Using two or more UV layers allows to combine different wall textures with windows and doors, making the resulting district much more varied. The mega-texture is created automatically from textures deposited in a specific folder hierarchy. This makes it much easier to exchange or extend facade, window or door textures.
Another very important type of asset is the street sign. There are hundreds of different street signs and the road network has thousands of signs distributed along roads and at intersections. The generated sign geometry is essentially a quad primitive with the texture according to the sign. Creating each sign as a separate geometry would make the scene-graph explode, which is why all signs are merged into a single geometry, at least for a district. Here again the mega-texturing allows to put all sign textures into a single material. The differentiation of the signs in

the virtual environment is achieved through the specific UV coordinates corresponding to the road sign.

4.2.4 Small Assets and Nature Elements

A huge virtual scene like an open world environment is filled with medium and small sized objects that are only perceivable when close to them. Nonetheless it is crucial to find a strategy to optimize the scene-graph, because adding naively all those objects to the scene results in huge performance issues due to high polygon and object counts. The big amount of polygons will impact the rendering process, and the high object count will impact the graph traversal of the scene-graph prior to rendering a frame. The solution applied in this work is to use level of detail (LOD) nodes and construct a hierarchy based on a space partitioning octree. The result is a good optimization of the scene-graph as only the branch of the LOD tree structure has to be traversed and rendered based on the distance to the user. All objects that are added to the virtual environment are appended to the bottom of the LOD tree, thus only visible if close enough.
This octree is used not only for assets like street lamps or road signals, but also for nature elements like trees and bushes. The difference is that trees are also visible from quite far away, a tree LOD is necessary that can be seen from afar, but with a very low polygon count as a forest area may have dozens of trees in viewing range. First the tree models have to created. Instead of adding tree assets designed with a 3D modelling tool, the approach taken in this work is to generate the trees using a basic growing algorithm. The data model is the tree armature represented by OpenGL lines and a point-cloud for the tree crown. The trunk and branch visuals are generated in real-time with the rendering pipeline based on the armature and advanced tessellation, geometry and fragment shaders. The first step is to tessellate each line of the armature and output curves with a certain resolution that can be dynamically adjusted. The second step is to replace each line segment by a tapered box geometry using the geometry shader. The last step is to use the fragment shader to do a ray-cast through the box and draw a perfectly smooth conical cylinder instead of the box. A bit simpler is the generation of the leafs. Each point of the tree crown data is replaces by a sprite using a geometry shader. The sprites are oriented based on the vertex information. In the fragment shader the leaf texture is applied, using an alpha threshold to discard non leaf fragments. This system offers many advantages compared to imported static meshes. Due to the parametric nature of the asset it is easily regenerated. This allows for example to simulate growing trees or change the treetop according to seasons. The simple data model of the armature also allows to add dynamic effects like wind or even to cut branches. The downside is that it is not easily possible to import created models for the tree LOD. The LOD models have to be generated based on the tree model. To generate the tree LOD models, the most detailed generated tree models are once rendered to texture from three positions. Those positions are axis aligned and as far from the tree as necessary to fit it perfectly on the image

canvas. Each snapshot is done twice, once to create a color image and once to create an image with the color encoded surface normals. The result is a set of six textures of the tree model, from two sides and the top, each in both variants with diffuse colors and surface normals. Those textures are applied on simple sprites and combined in a special fragment shader to approximate the correct lightning of the tree LOD. To achieve this it is important to correctly decode the surface normal, depending on the orientation of the sprite in camera space. If the fragment is the viewed from the backside of the sprite, it is important to mirror the normal in the sprite plane.

4.2.5 Rendering, Lights and Shadows

Open world environments have specific requirements to the rendering pipeline. They require for example to render complex materials, many light sources and shadows. An advanced rendering pipeline that addresses those requirements is deferred shading. Deferred shading is essentially splitting up the geometry processing and the lightning calculation into two stages. As the lightning is computed based on buffered data in image space, it is much more efficient to add many light sources to the virtual scene. Another important feature are shadows, they add greatly to the realism of an open world scene. The downside of shadows is that they require a separate shadow pass which significantly impacts performance depending on scene complexity. Additionally they have to be limited to a certain volume of space due to the important size of open world environments.
OpenGL usually comes with a classical set of light types like for example point, spot or directional lights. They do lack the realism of real world light sources. There is a method that allows to add a realistic light distribution to point lights to greatly enhance the lightning in a virtual scene. Those light sources are described by photometric data, a set of data available from lamp manufacturers. The method used to transfer to the rendering pipeline is to pack the data in a texture and modulate the point light intensity according to the viewing angle.

4.3 CAD Virtualization Process

The first step when creating a virtual environment is to integrate the geometric assets and define their basic behaviour. The integration of the scene assets is the first important bottleneck. The assets are usually loaded from the file system, network resources or dynamically generated. Assets are created in 3D modelling tools and exported using a suitable exchange format. Exchange formats do exist in abundance, but they are rarely well implemented in the corresponding modelling software like MCAD or ECAD systems, most workflows do result in heavy losses of information, reducing the semantic richness of the original models. This chapter will focus on the virtualization of CAD models. The first step is to define the data exchange pipeline. The second step is to merge the heterogeneous data into a single consistent model. The last step is

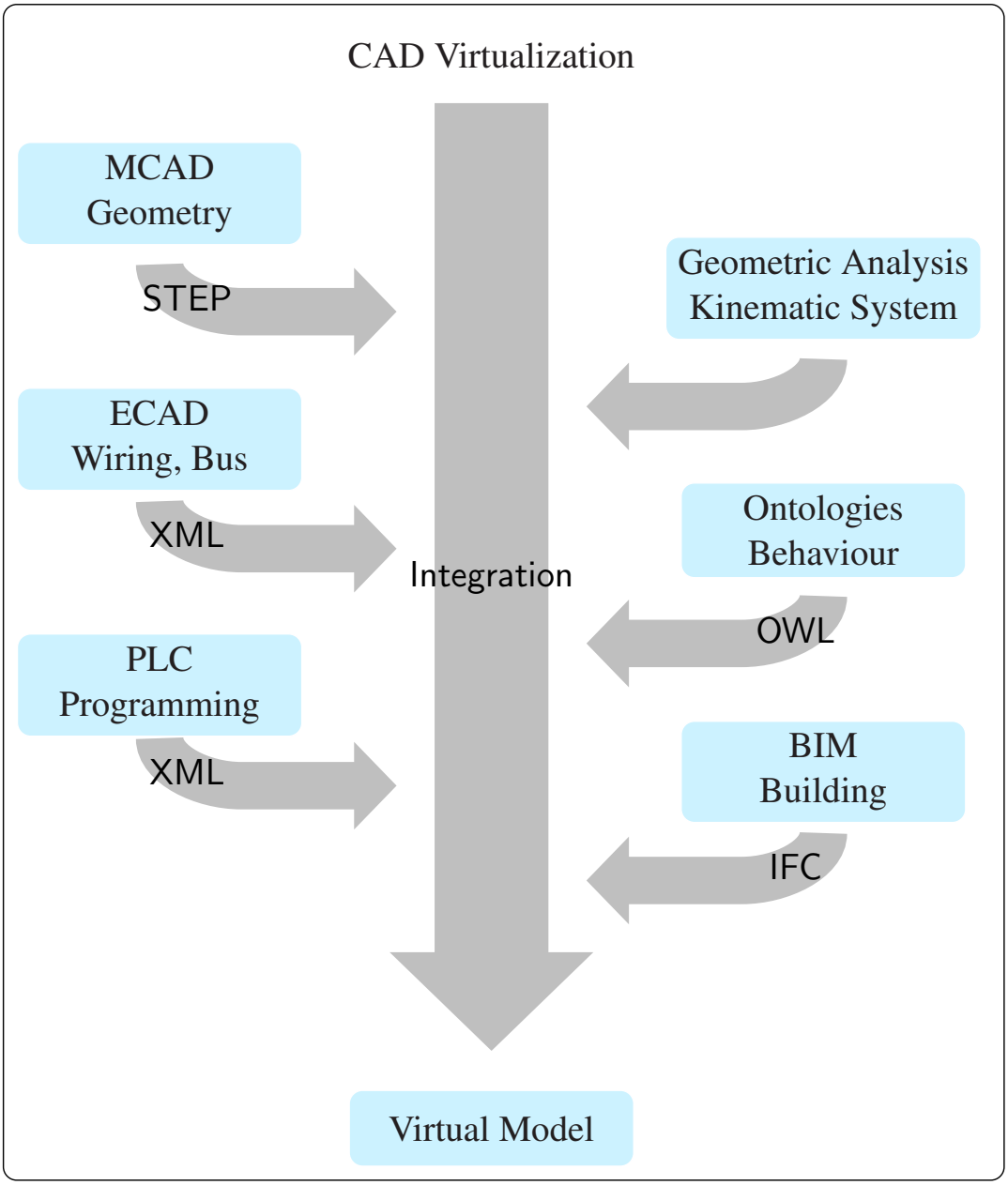


Fig. 4.5: CAD Virtualization Process

to enrich the model with semantic by analysing the data and add the behaviour to the model components. The goal is to obtain a fully functional virtual model based on CAD data. The process to achieve this should be as automated as possible. The concept is depicted on figure 4.5.

4.3.1 Mechanical CAD

The first set of workflows to discuss are the MCAD to VR workflows. They are the backbone of the virtualization process as they provide most of the geometric data. A high quality data foundation is very important to start integrating intelligence and growing the virtual model.

Data Exchange

The possible workflows to exchange CAD models can be classified as follows:

- triangulate CAD models and export them, for example as VRML97
- export the CAD models as BREP data like STEP or JT and tessellate in the VR environment
- extend the CAD software with a plugin and extract the tessellated models and send them directly to the VR application
- extend the CAD software with a plugin and send the original feature tree directly to the VR application

All those workflows have their benefits and downsides. The first one, using the tessellation from the CAD software and exporting using a standard exchange format, is the easiest as it works right out of the box. The downside is the complete lack of control regarding the quality of the exported data. Usually, most semantic information is stripped from the exported models, only keeping raw lists of triangles. Even the product structure and the part names are lost in the process, making this the least sustainable workflow. The second option is to use standard BREP exchange formats like STEP or JT to directly export the CAD models, then import the data in the VR environment and tessellate the BREP models for visualization. The advantage is a highly reduced file size as BREP is much more compact than triangle mesh approximations, furthermore most of the semantic data is also transmitted, including meta data, the product structure and much more. The downside is that the VR environment needs a CAD core module to manage and tessellate the BREP data. The next workflow is based on developing custom extensions, directly loaded in the CAD system. This allows a great variety of applications, even a bidirectional communication interface to send data back and forth between the CAD and VR systems. This allows to access all information, geometrical and others of the CAD system, tessellate the geometries and send them to the VR environment. This approach is very promising, but relies on the tessellation module of the CAD system, which can be a performance bottleneck. The last workflow is basically the same as the precedent, but it delays the tessellation to the VR system, again requiring a CAD core module to manage and tessellate the 3D models. Each of the workflows has its advantages, so they complement each other. Exchange formats allow to transfer data between all systems that support the corresponding format, and CAD plugins allow for a variety of functionalities based on live communication between the CAD and VR system, but they require a high development investment. Regarding the tessellation, there are two possibilities. Either the tessellation can be done in the design tool using the CAD API, or implement a tessellation module in the VR software. Implementing a tessellation module in the VR system allows to heavily optimize an important bottleneck in the virtualization

process. For example by using the GPU capabilities to greatly speed up key components of the tessellation algorithms. A middle ground is to use a library to include the tessellation functionality in VR. The strategic goal is to provide all those work-flows to achieve a high flexibility regarding the data integration for virtual environments. The data import like STEP does require the tessellation in the VR system, but when using a plugin in the CAD software it is possible to use the tessellation of those tools instead. A strategic long-term goal should be to implement a high performance tessellation core for the VR system, further reducing virtualization times, especially in regard to the ever growing complexity and size of 3D modelled machines, plants, or buildings.

Kinematic System

When designing mechanical constructions, it is possible to add dynamic information like constraints, kinematics or physical properties. The problem is that this is often not used or used as a constructive tool to help design the parts and assembly instead of building a consistent kinematic system. This makes the dynamic information too unreliable to use it for the virtualisation process. The method chosen to work around this limiting factor is to use geometric analysis to reverse engineer the parameterization of kinematic elements, gears, screw threads, axis, chains, levers, joints and more. This process is supported by a classification of the parts and an ontological knowledge base for the mechanical domain. The basic idea of this method is to simulate and automate the reading and interpretation of the CAD model as would do a human.

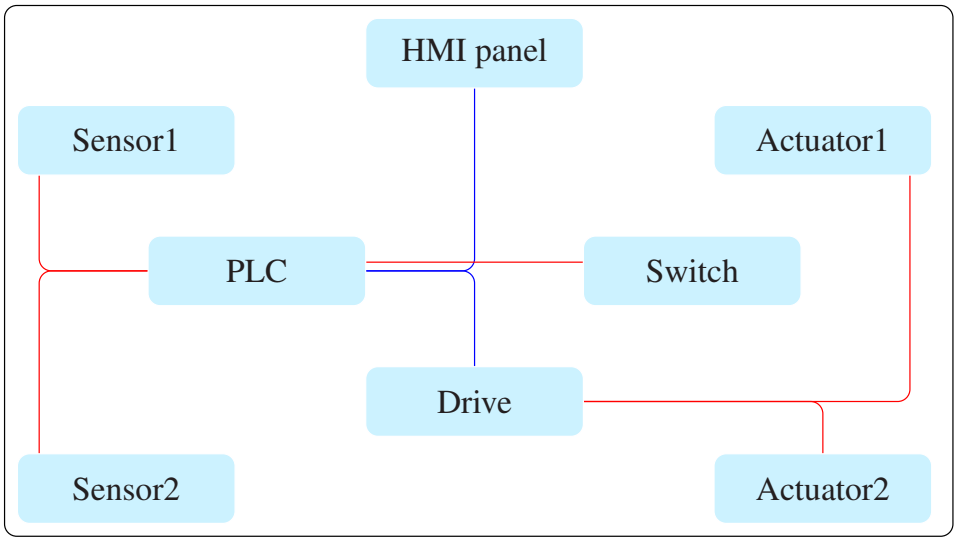


Fig. 4.6: Typical wiring with HMI elements like panel and switch, actuators and sensors, and the computational unit, the PLC. In red the electrical wires and in blue the bus system.

4.3.2 Electrical CAD

To further enhance the virtual model, it is necessary to access ECAD data, to map it to the MCAD data. It is important to understand the scopes of each, what the MCAD data can provide to the model, and where the ECAD and MCAD data overlap. MCAD data contains the geometrical information with a focus on mechanical and structural elements, with static and dynamic information like constraints and kinematic chains. ECAD data can also contain geometric information, especially for the representation of electrical elements, circuits, sensors, cables, electronics and much more, but it also contains all information on the electrical circuits, electronics components and bus systems. This information describes the connection between actuators, sensors and computational units, this is the basis to extend the virtual geometric models with functional behaviour.
The information of the wiring is the most important part of the ECAD data. A typical example is shown on figure 4.6. It is structured as a planar graph where each node is a ECAD component and each connection is a wire, usually a power cable or a data cable. Using this information allows to simulate the whole wiring system and the bus communication between components.

4.3.3 PLC Programming

The last data source needed to achieve a functional virtual model is the programming of the computational units. Similar to MCAD and ECAD design, the programming of the PLCs is

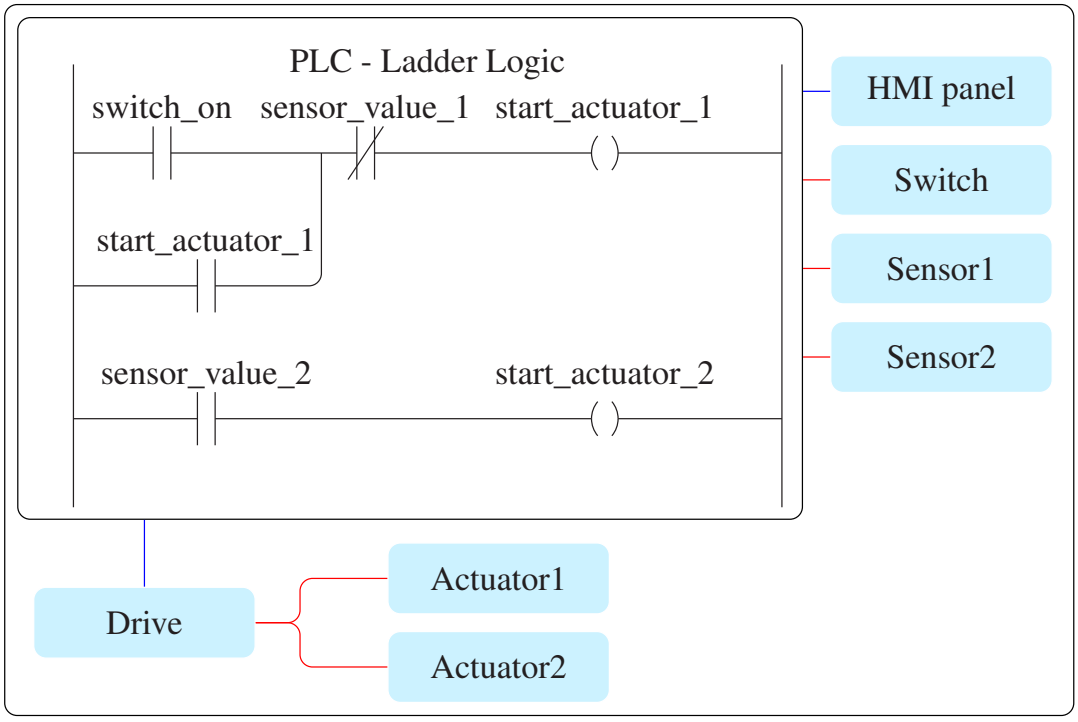


Fig. 4.7: Ladder logic example based on the wiring depicted on figure 4.6

a separate domain, requiring its own software tool and expertise. The aim is to program the behaviour of the PLCs. This is usually a sequential control logic, that defines the sequential execution of a process or manufacturing operation. A PLC is connected to HMI components like switches or panels through electric wires or the communication bus system. Those components allow the user to change the state of the system by setting variables. A switch for example will close a circuit which will change the state of a variable in the programmed logic of the PLC. The goal is often to compute the state of actuators, the speed of an engine or position of a joint. The corresponding variable is set and used by for example an engine controller to power the attached actuator. The program is directly exported to the PLCs of the machines. This means that it has to be working and complete. The combination of the wiring information from the ECAD data and the programming data for the PLCs allows to create the functional behaviour of a machine or production line. To achieve this the code has to be interpreted and executed in an emulation tool.
A typical programming language used to describe the control logic is the LADDER logic language (LAD). It is a graphical script language, historically based on the schematics that describe the relay racks as used in manufacturing and process control. There are other languages but this work focuses on LAD due to the available validation use-cases. It is easily possible to extend the virtualization system with other emulation modules for those languages if necessary. Figure 4.7 shows an LAD example for the wiring described on figure 4.6. This simple logic evaluates the switch and the sensors and controls the actuators accordingly.

4.3.4 Building Information Model

When planning a new production facility it is necessary to take into consideration the context the project will be build. A production line is usually inside a building, a new building planned alongside the production plant, or an old building that has to be used as production site. Building CAD models, similar to the CAD data of machinery, have geometric data but also the wiring and programming of building automation systems. Further planning data are heating, cooling and venting systems as well as the whole water infrastructure and much more.
Historically, CAD building data is two dimensional and contains only geometry without semantic information. A modern approach is to use semantically enriched CAD building models. Those models contain much more information that greatly adds to the virtual model. The geometric modelling is more and more available as fully three dimensional design. In addition to the geometry, the parts have attached to them meta-data like material properties, functional properties and product properties. The established exchange format for such data is the Industry Foundation Classes (IFC) data model for construction industry and building data. It is an EXPRESS based CAD Format defined in ISO 16739:2013.

The aim of this thesis is to support IFC for the import of BIM data and integrate the building as fully detailed virtual model. This allows to further plan and validate the CAD planning data by taking a broader context into account. When planning the production layout, collisions between the machinery and building elements can be detected and resolved, especially in 3D. Every boundary condition can be analysed like the access to building resources or complex interactions between the building and the production line like intralogistics. The material flow is a central aspect of a production line and has to be designed while taking many boundary conditions into account. The material flow spans the whole production line as well as the building. Just in time logistics have to be planned and optimized according to building geometry and production layout. The material flow is also important between different parts of the production line. The first part of the production may be the processing of raw materials, the second part may be the assembly of the product followed by the last part, the packaging and palletizing. There are many interfaces between engineering teams that have to be validated, a functional virtual model can greatly help to achieve this.

4.3.5 Assembling the Model - Fusing the CAD Data

The data sources for virtualizing industrial plants as described above are the mechanical and electrical CAD data, the programming and the BIM data. The problem now is how to integrate all that data in a single virtual model, in a sustainable and reusable way, maybe even fully automated. There are different approaches to MCAD and ECAD construction and product development workflows overall. A key aspect is the company specific way the CAD data is managed, how the components are named in each system and referenced between systems. The goal is usually to develop and plan with CAD tools up to the point where the data is used to build the physical product. This is often not compatible with directly reusing the data to easily create a consistent virtual model.

The first approach taken in this thesis to tackle this issue is to analyse the workflows in a company and to create customized virtualization workflows, embedded as best as possible in the product development process of the company. This approach is usually limited to virtualize products in a small scope and with a lot of handwork, thus very costly and time consuming. This thesis proposes an approach to use a semantic layer to abstract the whole virtualization process and thus allowing to fully automate it, at least in a predefined scope like plant engineering. The system should be able to handle high amounts of data, allowing the virtualization of huge production lines. The first step is to aggregate all CAD data as described above, and then enable the virtual reality system to infuse the data with intelligent behaviour. This is equivalent to automate the classical application logic development, and will be discussed in the next section.

Semantic Layer

When implementing artificial intelligence, there are mainly two approaches. The first one is a bottom-up approach, using machine learning based on neural networks or other empirical methods. Neural networks work well when used for pattern recognition or similar. One downside is the difficulty to build a system that can handle a broad range of different situations, where complex deductive capabilities are required. This is especially true when the system has to find out how to behave in a situation, where there is no provably correct answer, thus requiring basic reasoning and using uncertain knowledge. Training a neural network or similar machine learning methods are not suitable to build such a system.
A promising alternative for artificial intelligence is to use a descriptive method. The method focused on in this thesis is borrowed from semantic web technologies, domain ontologies and reasoning, used in virtual environments. Instead of training a system using labelled data, generic knowledge is explicitly given in a machine and human readable form. This allows to model complex knowledge, with many concepts bundled into domain specific taxonomies. Explicit knowledge is defined as rules, more or less atomic axioms that allow the system to reason, to make deductions.
The data integration requires to build up generic knowledge in a machine and human readable form. Such a knowledge base defines explicit knowledge in form of a taxonomy and rules, an ontology. Paired with a reasoning system, this allows to read heterogeneous data and use deduction algorithms to fuse it in a coherent representation. The theoretical background on ontologies is described in chapter 2.4. The limiting factor would be the generic knowledge available to the system, but this will be treated as a boundary condition in this work. The necessary ontology domains for the data integration are many, but the higher level domains are MCAD and ECAD as well as BIM and Automation. Those will be developed in the scope of this thesis and validated with various examples. A simple example is depicted on figure 4.8.
The generic knowledge defined in the form of ontologies is the basis for the generation of the functional behaviour of machinery. To be able to use that knowledge it is necessary to create a semantic layer. A semantic representation of the model, based on the CAD data, using the classification of the parts and components to map them to generic concepts in the ontology. The result is a set of instances in the ontology that form the semantic layer. As each concept has properties it is important to gather as much information and meta data from the CAD data and integrate it as properties for each instance. This is called populating the ontology and allows to process complex queries on the now complete knowledge base.
The MCAD data contains not much meta information. To populate the ontology with MCAD components requires analysing the geometry to compute the properties. When for example a component is classified as a gear, it is important to compute the properties that are relevant for parameterizing the part in a mechanism simulation. This example is shown on figure 4.9 The

ECAD data contains a great amount of meta data that can directly be included when populating the components. This is important for simulating the wiring later on.

4.3.6 Functional Simulation

The integration of MCAD, ECAD, BIM and automation data should result in a consistent model that represents the development progress at each moment in the product development process. This is a good basis to add behaviour to the model. The behaviour is the aspect of reacting to user interaction, beyond simple navigation and exploration. Machinery has many elements with dynamic and functional properties like actuators, mechanisms, joints or programming. This behaviour is usually implemented using various features of the 3D engine used to visualize the virtual mock-up, because the behaviour usually induces visible changes in the virtual model, thus changes to the scene graph. Most 3D engines provide scripting environments, easing the development of the behaviour, or application logic, but this still means a slow, time consuming,

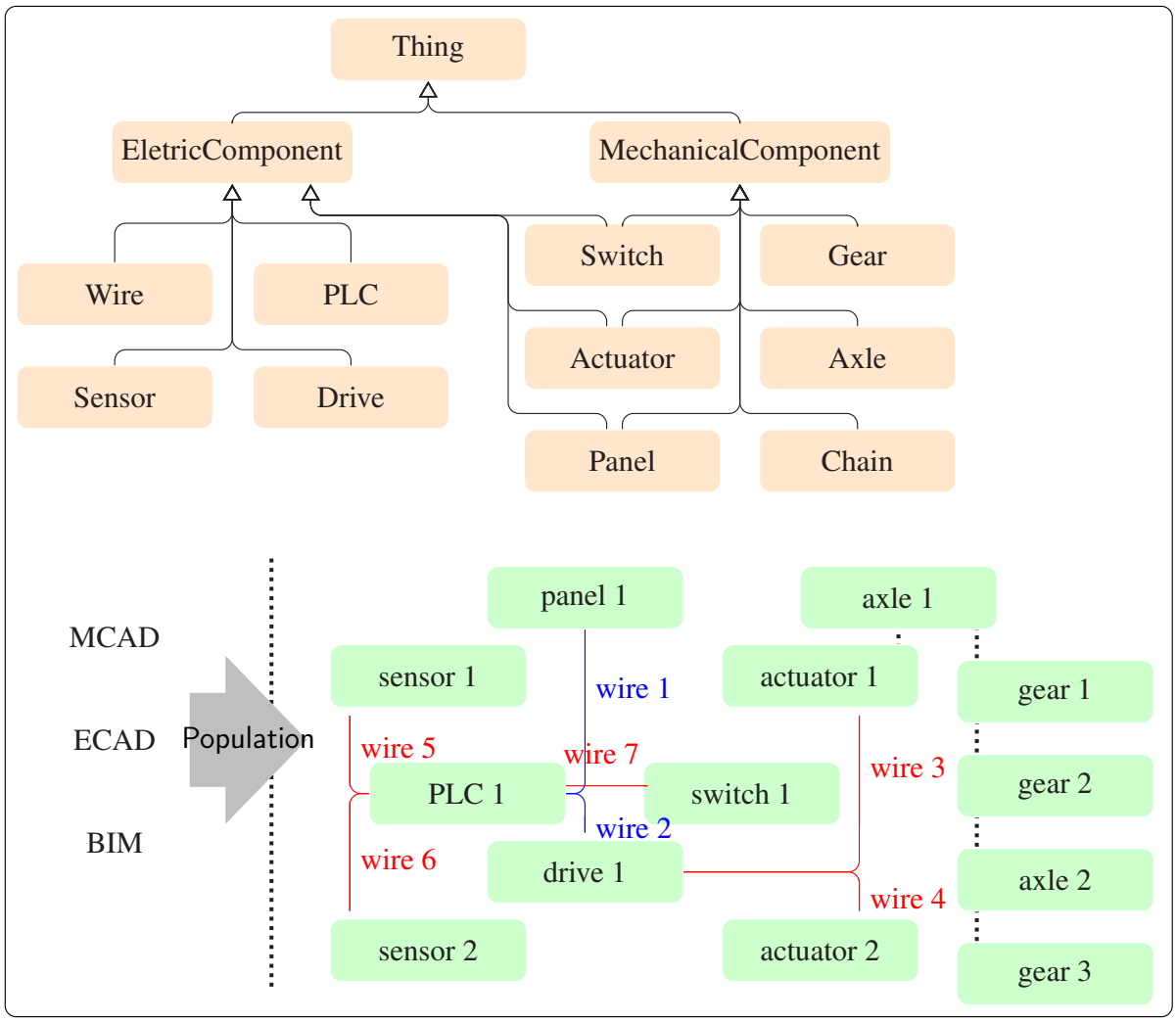


Fig. 4.8: Simple example of a semantic layer with MCAD and ECAD components, the taxonomy and below the instances.

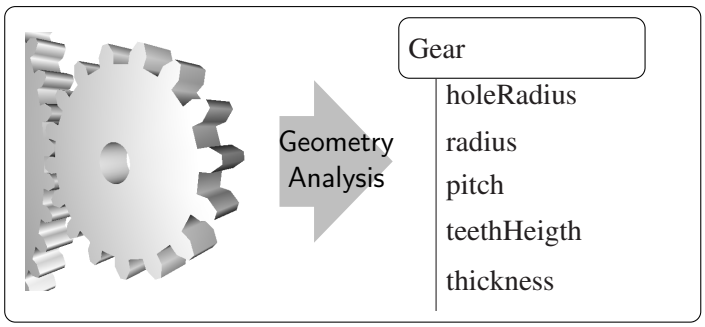


Fig. 4.9: Analysing geometry to compute part properties of a gear.

manual virtualization workflow. To avoid this, it is important to automate the creation of application logic, or at least automate complex application behaviour. This is essentially infusing the system with some kind of intelligence, the capability to solve complex tasks, answer queries and ultimately simulate a virtual environment that reacts intelligently to the user interaction.

Wiring Simulation

To achieve this it is necessary to add artificial intelligence functionality to the virtual reality environment as described in the previous section. The method used is to describe generic knowledge using ontologies and create a semantic layer containing all the knowledge of the CAD data and virtual machine model. The first use of this semantic layer is to simulate the electric wiring. The wiring data model is a graph, where each node is an electrical component. Those components are connected through electric wires. Each component has a specific behaviour that defines the outgoing currents depending on currents going in. Next to the electric wires are data wires like the bus system. the components that are connected on the bus system are programmable computation units that can compute and exchange information. This shows how the electric system interacts with the programming of the PLCs. Variables are set depending on the electric signal on the inputs of the automation module. The programming logic computes the changes to other variables, controlling actuators by outputting current on the electric wiring.
To simulate the wiring it is necessary to have all components with their properties in the semantic layer. Especially the topology of the wiring as well as the rules that define the behaviour of each component are important. Once defined, the reasoning system can be used to deduce the changes of the system from one time frame to the other. This allows to emulate the behaviour off all electric components. First an ontology has to be defined. Some key concepts as shown on figure 4.8 are for example the wire, the PLC or the switch. The simulation method will be explained based on the minimal example of a switch attached to the PLC as depicted on 4.10. The image also shows part of the taxonomy of the ontology used to support the virtualization of the wiring. Each electrical component has ports, labeled sockets where power and data cables are plugged in. This is also the main information from the ECAD data, the components and

the wiring topology. To simulate the wiring it is necessary to follow the current, from the main power line to each component. If the electric current reaches a port of a component, the type of component will change the state of its other ports. This can be for example a simple clamp or fuse which under normal circumstances lets the electric current pass. More precisely, if there is some current at one of the two component ports, the other port will have the same current. One level more complex is a switch component, its behaviour depends on its state, the current passes from one port to the other only if the switch is pressed. Every electric component has its generically defined behaviour. Programmable automation modules have the most complex behaviour. They have many IO ports that can be extended with more as needed. The behaviour is defined by the programming of those modules but from the perspective of the wiring simulation they essentially apply a current or not on their output ports. The simulation workflow is as follows:

every tick, execute the following process:

- reset all currents, ports and wires

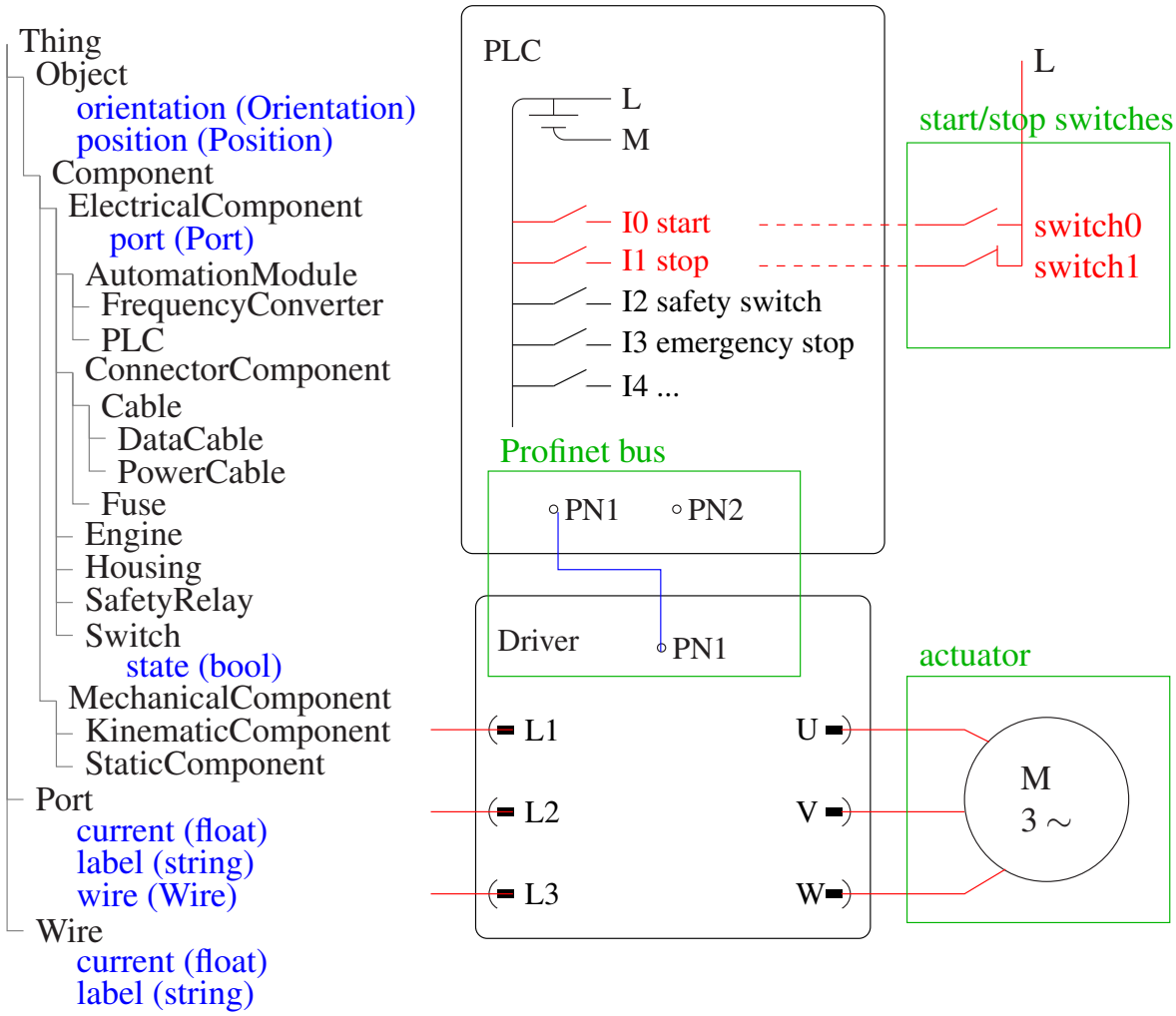


Fig. 4.10: Minimal wiring example, PLC with actuator, start and stop switches. Taxonomy on the right.

- set current at main power line
- traverse wiring graph

  for a **wire**:

  - transmit current from input port to output port

  for a **component**:

  - check its ports
  - compute the currents at its other ports

LAD Emulation

The automation modules like PLCs and motor drivers play a central role in simulating the functional behaviour of the machinery. They interface with the wiring by analog and digital input and output modules. The program that is running on those modules define a set of variables used throughout the application logic. Those variables are shared between the modules on the bus system. The electric wiring can change the value of specific variables by a changing the current on an IO socket of the PLC module. That way a switch can be toggled which results in a variable value change on the PLC to register the state of the switch. The program can now take this event into account and compute the impact on other variables. Once computed, those variables can result in a change of current in the electric wiring, for example to control an actuator.
To simulate this it is necessary to parse the PLC program and emulate its execution. The programming language that is analysed in this work is the LAD language. The program is structured in so called computational units which themselves contain a small graph with operators and function blocks as depicted on fig. 4.7. The simulation of this type of code involves traversing the graphs once per simulation tick. To traverse the graph of a computational unit one has to first setup an evaluation queue. This queue has to be carefully set up taking the graph structure into account. There are two ways to traverse the graphs, one is to first follow the connections, the other is to first evaluate the siblings. The correct traversal is to prioritize the siblings because nodes where two or more branches merge together need to know the values of all previous nodes. The traversal is depicted on figure 4.11. The algorithm to create the traversal queue goes as follows. Start with the power rail node, the line on the left, and put its child nodes on a stack. Then iterate over the children and add their respective children on the stack. Continue until all nodes are traversed. This will result in some nodes being added multiple times to the stack, just remove the duplicates and keep the top most occurrence. This will ensure that for all nodes, their preceding nodes will be processed first.

4.3.7 Mechanics Simulation

The simulation of the wiring and the emulation of the PLC program logic allows to simulate the behaviour of all the electric components in the virtual environment. The user interaction with HMI elements changes the state of the components up to the actuators. This allows for example to start an engine. The next step is to simulate the kinematic chain that is attached to the actuator. As described above, it is not advisable to rely on dynamic information from MCAD data as ti is often lacking or of poor quality. The method proposed in this work is to analyse the MCAD geometry to recreate the kinematic chains, extract the values to parameterize the mechanisms and simulate the dynamic behaviour of kinematic elements and mechanisms.

Geometric Analysis

The basis for a kinematics simulation is the description of the dynamic components of the mechanical CAD parts. The classification of the parts is given, this is the starting condition of this method. The goal is to analyse the geometry of the parts to extract various parameters and features needed for the kinematic simulation.
An example is the analysis of gears. A gear has typical parameters like a hole radius, a thickness, the number of teethes as well as their size, and the pitch, the distance between teethes. To compute those parameters, it is important to figure out the main orientation of the gear, the

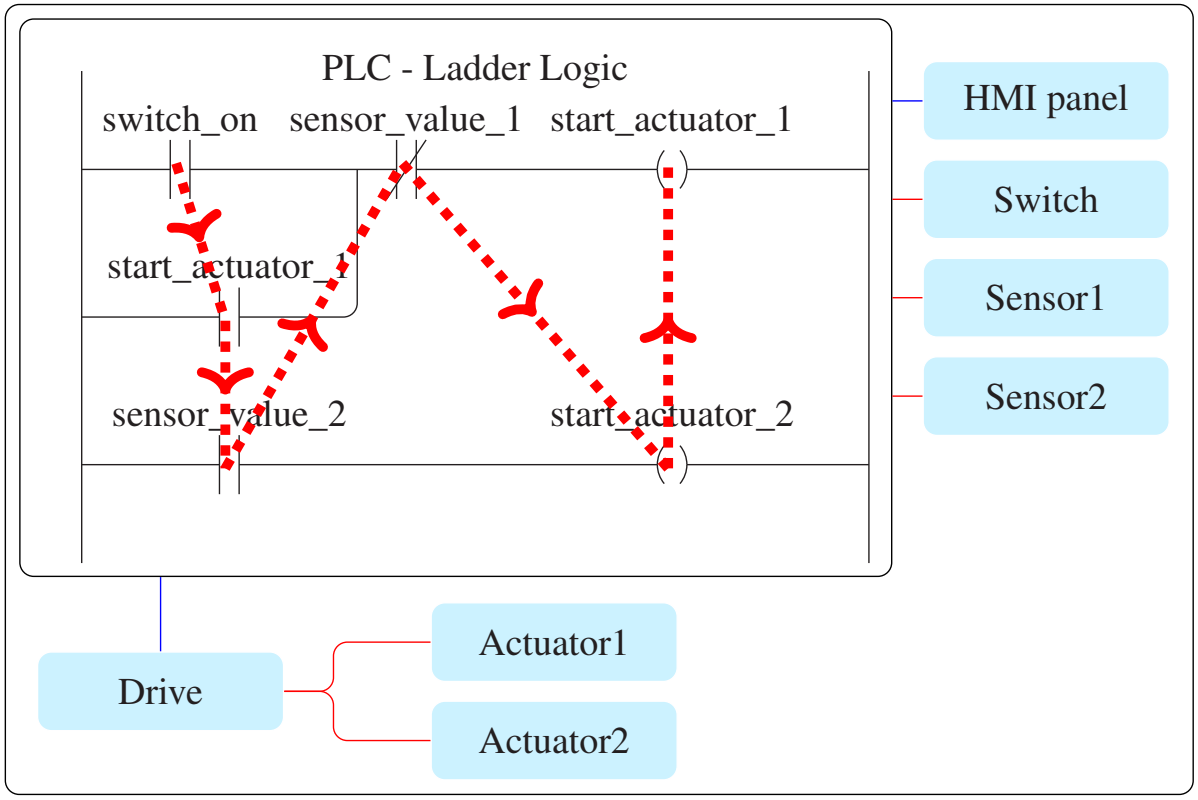


Fig. 4.11: Ladder logic example with traversal order, prioritizing the siblings.

rotation plane. The PCA algorithm allows to compute the distribution in space of the gear mesh vertices and yields the direction of the rotation axis as seen on fig. 4.12. Further analysis can be done in 2D by projecting the vertices in the rotation plane. A coordinate system has to be spanned based on the rotation axis and an arbitrary vertex defined as origin vertex. This allows to get the first two parameters of the gear, the inner radius of the hole and the outer radius of the gear. The next step is to compute the polar coordinates of the vertices and sort them according to their angular coordinate. The result is a discrete function that resembles a sine wave as seen in figure 4.13. Once re-sampled to get equally distanced points one can apply an FFT to get the main frequency of the periodic structure of the function, the gear teethes. The inverse of the frequency is the angular pitch of the gear teethes, multiplying with the respective radii yields all the different pitch values of the gear. The last step is to fit a sine curve (fig. 4.13) to get the last parameters like the amplitude that corresponds to the teeth height as well as the vertical offset that corresponds to the mid-teethes radius. All those parameters can now be used to parameterize the mechanism simulation.

Kinematic Simulation

The kinematic simulation is necessary to add basic dynamic behaviour to mechanical parts. The simulation handles the motion of parts taking into account the constraints imposed by collisions with other parts and mechanical joints. The physics engine used in PolyVR do provide such features. the kinematics simulation does have special requirements that differentiate it from directly using the physics engine. The main difference is that the kinematics simulation does not represent the effects of gravity or inertia. The goal is to be able to simulate and interact with machinery without the machine parts falling apart. The user should be able to move a lever in

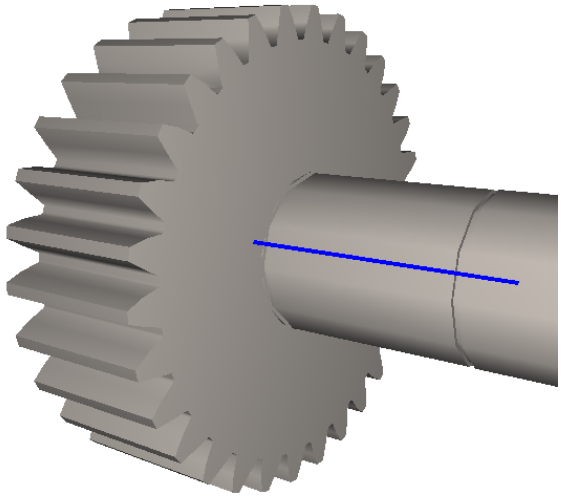


Fig. 4.12: Use PCA to analyse a gear geometry and compute its rotation axis.

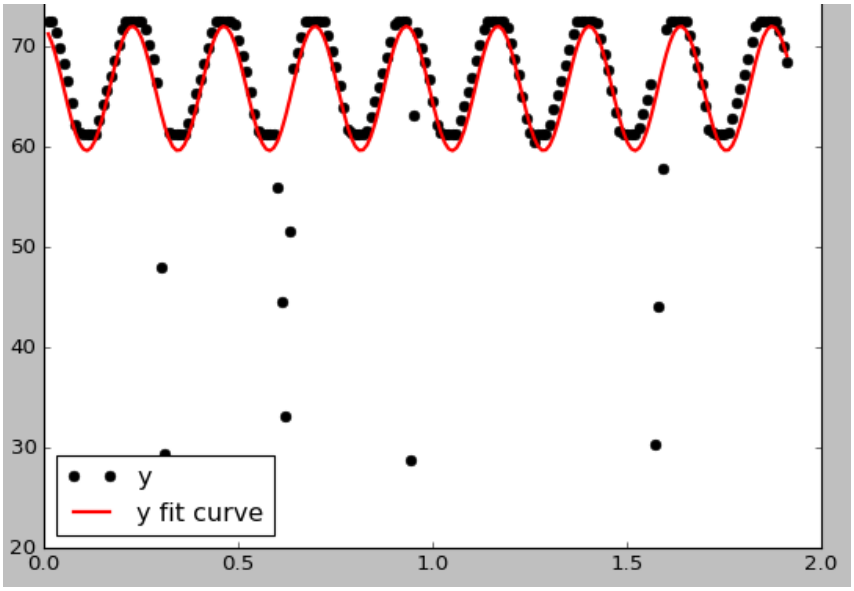


Fig. 4.13: Analysis of a gear geometry, fitting a sine curve to the vertices of the gear teethes.

a certain position without the lever dangling back and forth. This can be achieved by parameterizing the physicalized components to ignore the effects of gravity and set a high damping for linear and angular motion. Using the physics engine as a basis eases the combination of the kinematics simulation with with the rest of the physicalized environment. It also simplifies the definition of the kinematic data model as it is the same data basis. If the user wants to define a kinematic chain he can pass it to the regular physics engine or pass it the kinematics simulation. The most important joints are the ball, hinge and fixed joints. They are specific variants of the 6 DoF constraint which allows the user to lock or free any degree of freedom, along the translation or rotation axis, and even limit them to a certain range.

Mechanism Simulation

The simulation of mechanisms is primarily used to handle the interaction between gears, screw threads and chains. The kinematic simulation as described above is not suited for handling efficiently this kind of connection between objects, because a kinematic simulation does primarily handle collisions and 6 DoF constraints. The simulation of a mechanism has to transmit a change of its parts to their neighboring parts. The possible combinations are gear-gear, gear-chain and gear-thread. Other connections like gear axis and other kinematic joints attached to the mechanism parts are not handled by the mechanism simulation. The connection between a gear and an axis is a fixed joint, it is thus simulated by the kinematics simulation. Both simulations work together to simulate the entire mechanism. The mechanism simulation work flow is as follows:

every tick, execute the following process:

- check all parts for transformation changes
- for each changed part:
    + update its neighbors
    + compute its effect on its neighbors
    + propagate the effect to its neighbors
    + check if the propagation succeeded, if not then reset to the old transformation
- apply the new transformation of each part to its geometry

The way the simulation workflow is designed makes it very flexible, especially due to the ability to react to any transformation changes of its parts. The part transformation changes can be induced by another simulation like the physics engine or the kinematic simulation, or simply by the user. The transformation change has to be processed to yield a value that describes the impact of the transformed part on its neighbors. In the case of a gear the effect of the gear is mainly based on the rotation component of the transformation. The effect transmitted to its neighboring gear is the rotation angle multiplied by the gear radius. The neighboring gears get this value, divide it by their own radius and get the rotation angle they have to apply on themselves. When a chain is attached to a gear, the value gets propagated to all other gears attached to the chain. It may happen that a part receives two values during the same propagation traversal but from two different neighbors. If this happens the values have to be checked and if they do not result in the same transformation the whole propagation fails as the mechanism is blocking.

4.3.8 Processes

Once the virtual model is fully functional, it is ready to execute more complex processes. This is more an outlook as it has not yet been fully implemented in PolyVR. The first step is to import a process description. As intuitiveness is a key aspect in virtual reality applications, the efficiency and intuitiveness of modelling processes is very important. Elstermann et al. [EO14a, EO14b, EHH17, HEHO16] propose an extension to the subject oriented process modelling as well as the combination with immersive environments for collaborative modelling and validating of processes. The subject oriented process model does synergizes nicely with an agent based simulation as often found in a typical virtual environment.

4.3.9 Communication Interfaces

An aspect often addressed in the context of the virtual twin is the combination of the virtual model of a machine with its real counter part, resulting in a so called cyber-physical system. This is usually achieved by incorporating sensor data in the virtual model or mimicking the real behavior of the machine. This allows for example for advanced applications like process planning, monitoring, and optimization.
The setup of such a cyber-physical system, at least on the software development side, is quite trivial once the virtual model has been created with a sufficient complexity and quality. The

communication with industrial systems like PLCs can be achieved through different communication systems. From a simple web socket over ROS to OPC UA and MQTT. PolyVR does provide an interface for setting up web socket communication or using the OPC UA protocol.

4.4 Virtual Engineering Applications

Virtual engineering encompasses the whole product life cycle and supports it with a variety of applications like the design review during the product development process or the production planning for manufacturing. In this work the focus lies on the virtualization and the applications that are mainly based on and derived of the virtualized model. This is as just mentioned the design review application, but more simple the viewing of models and data. A more advanced applications is the training of workers for the configuration, operation and maintenance of products, often machines or production plants.

4.4.1 Model and Data Viewer

The most basic use of the virtualization process with all its data interfaces is to visualize CAD models or other data like CAE simulation results or other scientific data. The user can load up the model and explore it in an immersive hardware setup. Such a viewer application does not have many interactive and functional features apart of basic navigation. On the other hand it is important to offer many data interfaces as for example for mesh formats, CAD assemblies or point clouds. A model viewer is also a show case application of a 3D engine, the rendering capabilities have to be displayed, but the user is not supposed to design the scene apart loading his model. This means that the default settings for such an application have to feature advanced effects like shadows and good lightning. The difficulty is to choose the default rendering settings and scene design in a way that fits most models.

4.4.2 Design Review

The design review application is essential to validate the CAD construction progress and product development overall. The degree of automation of the virtualization process is key to maximize the added value of the design review, especially when used regularly to validate the current development progress. The system has to enable the construction engineer to load up the virtual mock-up in just a few minutes. The more it can utilize the better. Enriching the data with semantic information also allows to visualise specific relations between parts or filter parts by domain to customize the view according to the users needs.
Such an application has to be packed with tools to explore the model interactively. Drag and drop allow to take the model apart, each part can be selected and hidden individually. Materials can be set to transparent and a clipping tool defines a half-space where the model is rendered,

but clipped on the other side. All those tools allow to go through the model layer by layer, discussing interfaces and construction spaces. The goal is to find potential issues and collisions and to resolve them before the first part gets produced in real. Such an iteration can take a whole day until the changes are introduced into the model and the engineers are back in the immersive hardware setup to further discuss the state of the development progress.
To further reduce the time for each development iteration one can add a CAD workplace next to the immersive environment. This allows to quickly correct problems that emerge while evaluating the model in VR. The resulting changes in the model can be synchronized on the fly to the virtual model. This makes the whole process even more interactive and efficient.

4.4.3 Training Simulation

A fully functional model of machinery is the perfect basis for creating a training simulation. The model allows to operate the human machine interfaces like panels and buttons, and change the state of actuators accordingly while virtual sensors change the indicator values, giving a realistic feedback to the user.

4.5 Summary of the Methodology

This chapter proposes a design concept for a virtual reality authoring system, that aims at greatly simplifying the creation of virtual environments for engineering applications and the deployment of virtual reality applications in high-end immersive hardware environments. A broad array of modules will allow to ease the development of advanced engineering applications, from the various data exchange and communication interfaces to the use of knowledge bases and reasoning.
The automation of the CAD virtualization process provide a major incentive for SMEs to deploy virtual engineering solutions. The proposed system will enable the use of VR for SMEs, without the need to build up know-how in VR technologies, especially software development [Häf17a].

5 Implementation - PolyVR System Design

To access the possibility to advanced VR system development and research, it is necessary to build a solid foundation. This foundation has to be a code base able to integrate all the major open source libraries, and act as a safe harbour for new developments. A place to integrate ideas, prototype new concepts and try out new paradigms. This chapter describes the virtual reality software PolyVR [Häf17b], developed in the scope of this thesis. It is the result of many years of developing virtual reality applications for engineering applications, especially mechanical engineering. The framework is the condensation of the results of many research and industry projects. Other important development drivers were the students works [HHO13, HHO14, HSD+14]. To make VR application development accessible to a broad range of students created the need for educational resources. An intuitive UI, scripting environment as well as documentation were necessary for supporting virtual reality practical courses as well as bachelor and master thesis. In the following, the software architecture and the different modules will be presented and discussed.
PolyVR is a software that has to combine many aspects into one. This is typical for a VR software, major components are for instance:

- Graphics accelerated 3D application
- Integrated development environment for real time applications
- Virtual reality hardware deployment system
- Content management and creation system
- Simulation and AI system
- Web server and browser

All those components and roles have to work together to enable complex VR applications. It has to be an authoring tool and virtual reality framework all in one. A VR authoring tool needs to address certain needs, for instance:

- Build the scene

  A scene graph to structure the scene assets

Various IO capabilities to import assets

Materials, lights and shadows configuration

Access to the rendering pipeline

- Support content generation

  Sound synthesis

  Geometry and texture generators

  open world generation

- Create application logic

  Scripting environment

  Simulation systems

The main components of the authoring system of PolyVR are the scripting editor and the scene graph explorer. This offers the greatest flexibility and access to all functionality within the framework. The lack of a 3D editing environment allows to always edit the application while it is running, no run and edit modes are necessary. This means that every change to the application logic will take immediate effect.
Behind the authoring system is the virtual reality core system, the framework to support VR hardware, data interfaces, simulations and engineering modules. It provides implementations of the basic interaction paradigms like drag and drop, orbit and fly through navigation. Data interfaces and management allows to import mesh data, point clouds, textures, sound, CAD data and much more. To add intelligent behaviour to virtual scenes, various simulation modules like a physics engine or reasoning based on a knowledge base are available. Advanced hardware configuration capabilities allow to deploy VR application on immersive hardware and interface to industrial standards like ROS or OPC UA, thus paving the way for deployment in SMEs.

5.1 Software Architecture

The software design follows the methodology outlined in chapter 4.
As depicted on fig. 5.1, the system has layers to abstract low level functionality in basic modules, combined into more complex modules in higher layers, thus keeping the overall system highly modular but also allowing efficient synergies between modules. This chapter will describe in detail every dependency and module from bottom to top, starting with the system dependencies up to the user interface.

5.1.1 Implementation Specifics

The code base is fully object oriented. Memory management is implemented through the consistent use of smart pointers. This is especially important regarding the interaction with the threading system of the scene graph library, as OpenSG uses redundant data structures called aspects in combination with change-lists to implement threading and clustering capabilities [Rei02]. The scripting environment, the Python C API, also uses ref counting and thus harmonizes well with the use of smart pointers.

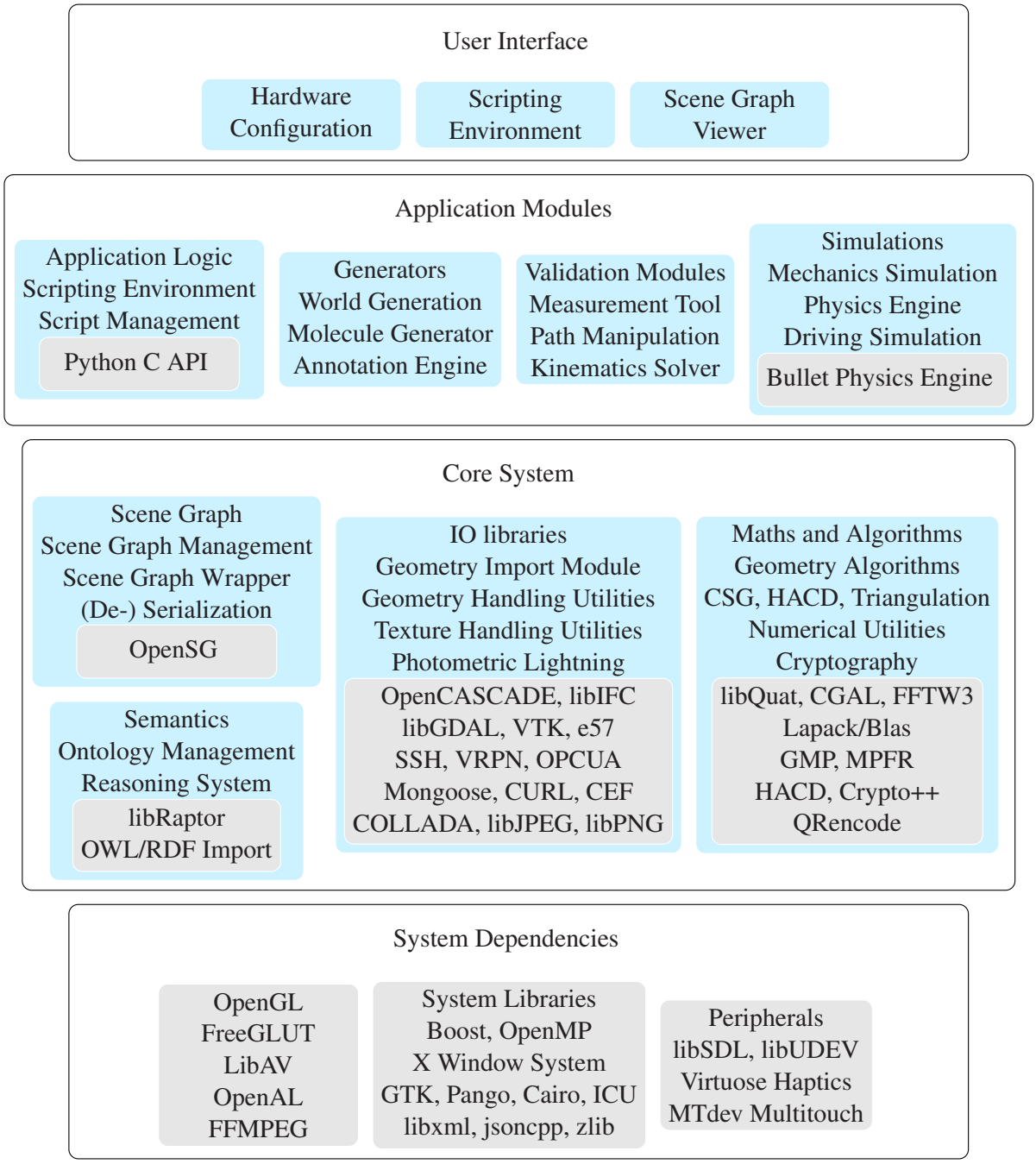


Fig. 5.1: PolyVR software architecture, emphasizing the software dependencies in grey

5.1.2 System Dependencies

PolyVR is a software that is very close to the hardware system and uses many low-level libraries to access a broad range of hardware resources. In the following, each major dependency will be briefly discussed.

Graphics Library

The graphics API OpenGL allows to access the graphics card, access to the default rendering pipeline, the shading computation and the GPGPU capabilities. The FreeGLUT library adds basic utilities around OpenGL, for example a basic window. OpenGL is portable, it is available for every system, UNIX based or Windows. It is the go-to standard when creating software with a 3D viewport. OpenGL by itself is very low-level, its main paradigm is that of a state machine, which is too bare bone when you want to create complex virtual worlds. This is where scene graph systems come into play, they wrap all OpenGL functionalities and expose them through an object oriented interface. For more information on the scene graph used in PolyVR see chapter 5.1.5.

Sound Library

Libav and OpenAL grant low-level access to the sound devices of the system. It is necessary to implement a sound system on top to ease for the user the handling of audio content, decode sound files, parameterize sound synthesis algorithms and manage sound channels. The management of for example the decoding of sound files or the . The system implemented in PolyVR adds basic sound channel management, feeding the sound card with sound wave packets, either decoded from sound files like .wav or .mp3 using FFMPEG, or packets generated from synthesizer. PolyVR has implemented multiple sound synthesizing algorithms, the basic Frequency modulation synthesis allows to easily generate a simple sound by parameterizing the carrier and the modulation waves. Another synthesis algorithm generates a wave based on a given frequency spectrum. By providing two spectra the system interpolates the sound wave between those two spectra over a specified duration using a phasor. The math behind the sound synthesis is explained in chapter 2.3. The various channels of the sound system can be addressed individually, this allows for example to provide vibration feedback to the user over the sound system and a special vibration subwoofer.

System Libraries

Other very low-level dependencies are the Boost libraries, a big collection of various utility libraries, from threading to file input/output. The OpenMP library eases the parallelization of specific code passages. The X Window System is the windowing system, common on Unix-like operating systems. It provides the basic framework for a GUI environment by drawing windows

on the display device and grants access to mouse and keyboard devices. On top of the X system, GTK is a portable, feature rich toolkit for creating graphical user interfaces. Pango, Cairo, ICU are libraries with various features around the graphical UI, like bitmap generation from Fonts, low-level bitmap drawing for UI widgets, or internationalization utilities. LibXML, JsonCpp and zlib are low-level libraries to handle files, read XML and JSON protocols and compress or decompress ASCII files.

Peripherals Libraries

The last set of low-level libraries discussed in this section are the libraries needed to access peripheral devices or VR hardware. LibSDL and LibUDEV provide APIs to access devices on the local system. The mtdev library allows to access multi-touch devices like multi-touch surfaces or multi-touch frames. When using mouse, multi-touch or similar devices, it is often necessary to configure the default X driver. It may for example be necessary to disable the default X input of the multi-touch device to avoid getting default touch events on the main desktop, and instead process the touch events in PolyVR. A simple way is to call the xinput utility program with according parameters. PolyVR has the necessary utility implemented that automates this easily. The last library is the Virtuose API, a library used to control haptic feedback devices from the company Haption GmbH. It is necessary to implement a module to wrap the Virtuose API and link with the physics engine to compute the forces needed for the haptic feedback, this has been done in PolyVR.

5.1.3 Scene Management

The top level structure of PolyVR splits into the hardware and scene management. This is based on the design choice to abstract the hardware as much as possible and decouple it from the development of the virtual worlds. Those two main modules are the (virtual) scene manager and (hardware) setup manager. For more information see the setup manager in chapter 5.4. The scene manager handles loading, starting and stopping of the scenes, and implements the top level application loop. When installing PolyVR, a folder with dozens of example scenes are available. Those examples provide a reference implementation for many modules, and allow to experiment easily with working scripts. This has proven very useful for getting a first glance at the broad spectrum of modules available in PolyVR, and greatly improves the learning curve when starting to develop virtual worlds.
Apart from managing the scene, the scene manager is the top most active module in PolyVR, as it is managing the main application loop. The manager has a threading and callback system that allows to easily manage and expose the main application loop and threads to other modules. It also includes a network manager to provide access to scene management functionality over

network interfaces. The scene management main loop calls its subsystems in the following order:

- Call GTK main loop (Run main UI event processing)
- Execute callbacks that have been registered for execution in the main loop.
- Update VR tracking and hardware device events
- Call scene update
- Trigger local and remote rendering
- Free Python GIL to allow user threads to run an iteration
- Increment the global frame count

This is fairly standard, noteworthy are the updates of the VR hardware, especially tracking systems, and the freeing of the python GIL. The GIL is a global lock that serializes the python execution. There is in this sense no real threading in python 2.7.

5.1.4 Mathematical Utilities and Data Structures

The scene graph library does provide basic data structures for linear algebra functionality, especially for homogeneous vector and matrix operations. PolyVR complements this with various data structures for geometric generation, space partitioning and more. B-splines and patches, a graph and an octree implementation, a polygon and triangulation module, state machine and other more specific functionalities. Other libraries providing basic mathematical functionalities are the CGAL library, Open CASCADE, Bullet and VTK.

5.1.5 Scene Graph

The scene graph is the backbone data structure of the virtual world. It is a tree structure whose nodes define the virtual objects. The structure defines the topology of those virtual objects. Depending on the 3D engine, there are slight variations of the structure and the nodes. PolyVR uses OpenSG as scene graph library, but heavily wraps the scene graph structure and nodes to extend it with many utility functionalities. There are different kinds of nodes, for example the basic object node, the transform and the geometry node. Fig. 5.2 shows the inheritance of the scene graph nodes and how each of them wraps the OpenSG nodes. Many modules inherit from scene graph nodes which allows to easily integrate them in the application by appending them somewhere in the scene graph. In the following, the scene graph nodes and the way they wrap the OpenSG nodes is described in detail.

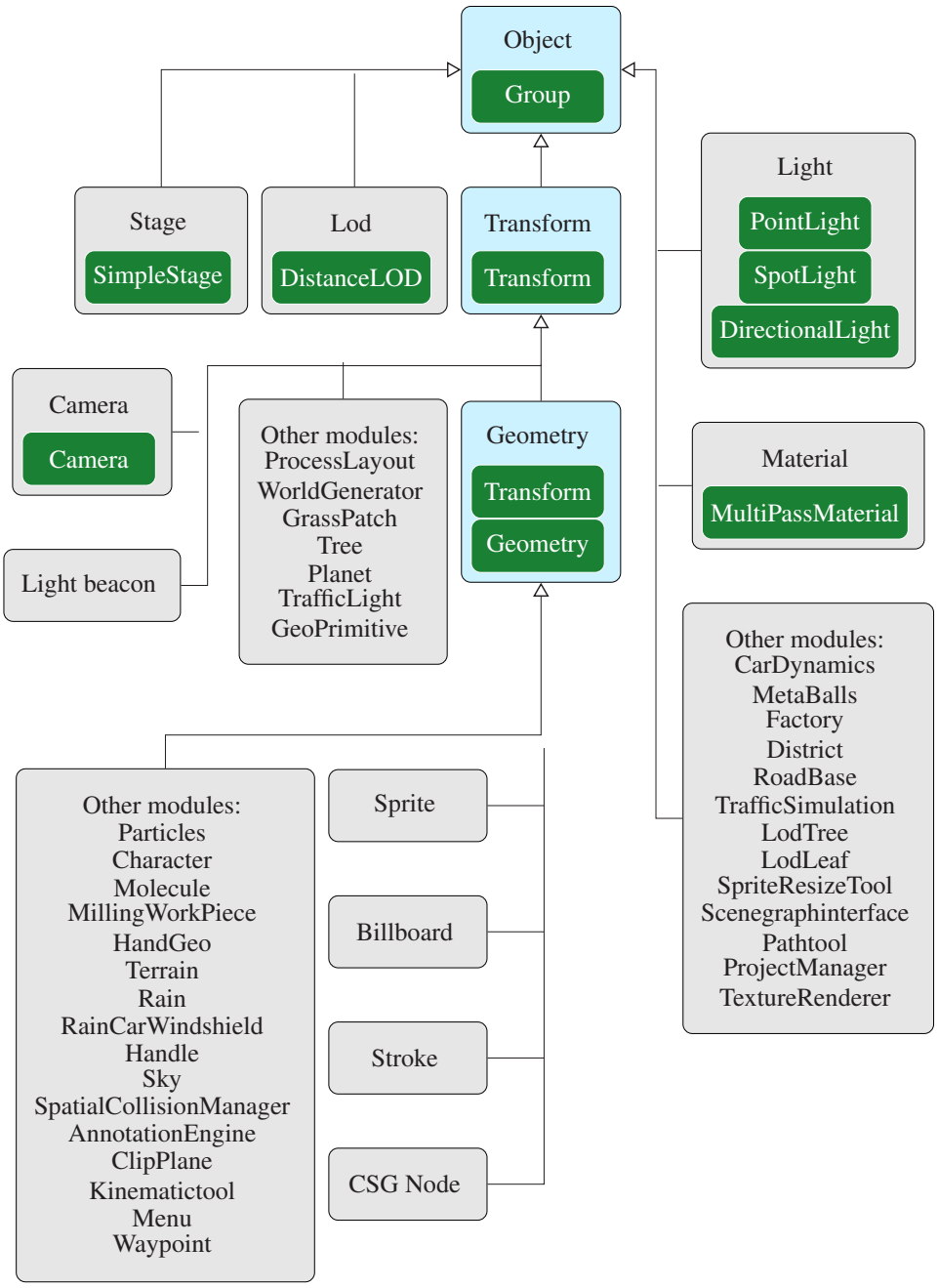


Fig. 5.2: Inheritance of main scene graph nodes, the wrapped OpenSG nodes are depicted in green, and the primary node types in blue.

The object node is a simple structural node that allows to group other nodes below it, forming the tree structure. Every node inherits from the object node. Another basic node is the transform node. It is a node that defines a pose in space, a position, orientation and scale. It spans a coordinate system and transforms its child nodes with their subtrees. The mathematical description of the transformation is essentially a homogeneous matrix.
The geometry node inherits from the transform node and adds a mesh. Meshes are a set of vertices in space, where each vertex has a position and further optional data like a surface normal, a color, or UV texture coordinates. The vertices are assembled to OpenGL primitives, points, lines, triangles or quads.
Light nodes are special nodes that define a light source to illuminate part of the scene, more specifically the subtree of the light node. When lighting a scene with multiple light sources, one can optimize the scene graph structure so as to limit the influence of each light to the objects surrounding it. Every light needs a light beacon, a transform node that define the light position in the scene. The beacons can be attached anywhere in the scene graph.
Camera nodes define a viewpoint in space, from which the system can render to its display viewports. It is again a node that inherits from the transform node, and extends it with camera parameters.

5.2 PolyVR Modules

PolyVR contains a modular toolset in order to facilitate the authoring of engineering applications. This chapter presents the most mature modules, describing their internal structure and applications.

5.2.1 Geometry Generators

Generating geometry based on parameterizable algorithms is an essential tool for quick prototyping of virtual content and synergizes well with a multitude of applications like visualizing scientific data or various information models, or even enabling objects with interactive and flexible geometry.
The basic geometric primitives available from the OpenSG library comprise the cube, sphere, cylinder, plane, cone, torus and the famous teapot. PolyVR defines addition primitives, the 3D arrow, the gear, and the screw thread. They are depicted on fig. 5.3.
One of the dependencies of PolyVR is the library for Computational Geometry Algorithms called CGAL [AF16]. It is packed with algorithms to deal with polygon and polyhedrons, including Boolean operations. An important modelling method in computer aided design is the constructive solid geometry. Geometrical primitives are combined using Boolean operators to create new geometry. The method is quite old and rarely used nowadays for actual mechanical design, but it can still be very useful in various cases. One such case is when prototyping virtual

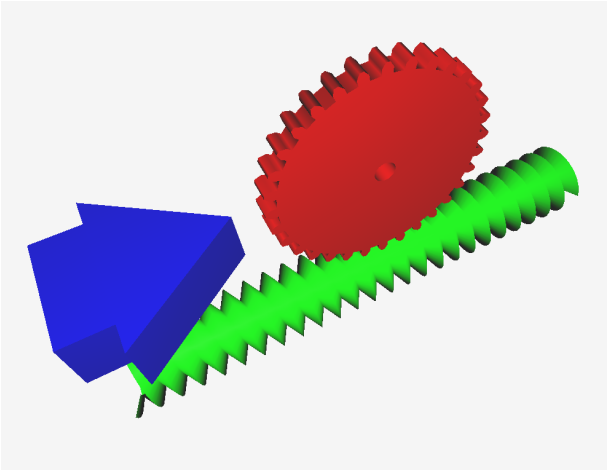


Fig. 5.3: Example of the use of parametric primitives in PolyVR.

scenes, it can drastically broaden the various geometries available to rapidly design a scene, in contrast to only having geometric primitives. It was also extended to work with sweep models that are also available in PolyVR. A good use case is the process of cutting materials with a laser or a water-beam cutting machine. The CAM program can be used to create a sweep model which can be deduced from the workpiece. Other use cases are to quickly add concave features to the ground plane or boring holes in a mechanical part.

5.2.2 Web Technologies

An important feature in virtual worlds are two dimensional widgets. They are useful for instance for displaying information, meta-data of virtual data models, or for adding control panels for production plants, or even just adding a classical flat menu, attached to the camera node.
It is important to facilitate the design and deployment of such widgets. For this purpose, a module was designed that allows to put any kind 2D interactive widget on any 3D surface. It achieves this by using web technologies, interactive websites that can be deployed as OpenGL textures. This means that the website can be rendered on any surface model. The module architecture can be seen in diagram 5.4, the core component for the rendering of the websites is the chromium embedded framework library [Gre09] and the core component for hosting websites is the mongoose web server [Ham10]. The main contributions of this work are the authoring work flow as well as the interaction loop integration.
The authoring of websites is supported with an embedded editor where the user can write any textual resource necessary for the website, usually using HTML, CSS and JavaScript. Any change can be seamlessly applied to the current scene with a button press, without restarting the scene or impacting it in any way. The back-end system is none other than the python scripting module described in 5.2.4. User scripts can be triggered by web server events, usually via WebSocket. The website has to open its own WebSocket and define callbacks for receiving and

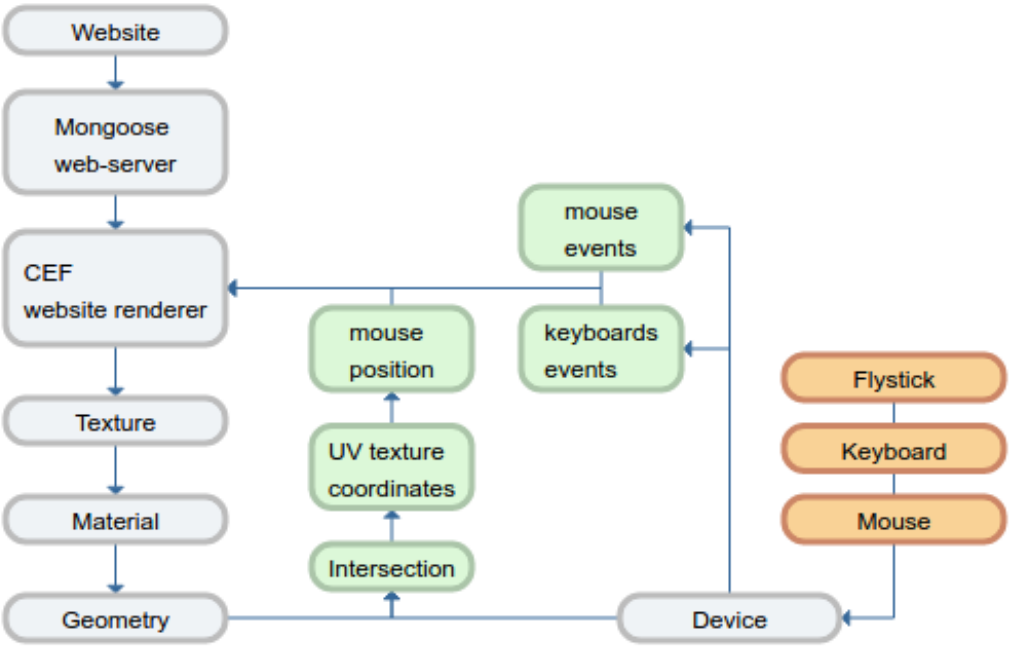


Fig. 5.4: Web widget module, architecture

sending data. A minimal implementation is done in three lines of code, it is documented with reference implementations in the examples available in PolyVR.
The interaction loop as depicted in fig 5.4 was conceived and implemented in a way to abstract the input devices like mouse and Flystick. The interaction with the website works out of the box. Any hardware device, virtual reality devices as well as desktop mouses, have a beacon, a transform node. Any interaction is based on the concept of casting a ray from the beacon into the virtual scene and intersect geometries. The ray cast uses the device beacon transform and intersecting the geometry yields the intersected triangle as well as the interpolated UV coordinates of the triangle vertices. The UV coordinates can be directly translated in mouse coordinates and fed to the chromium embedded framework. Device events like button clicks can be mapped to the mouse click events:

- Left mouse button
- Middle mouse button
- Right mouse button
- Scroll up
- Scroll down

Any VR device can thus interact with websites in a completely generic way. Apart from the core functionalities described above, the module has other interesting uses. The websites are

Fig. 5.5: Example of the use of photometric lights for more realistic streetlights.

hosted by PolyVR. Hence the user can open them in any browser. This enables to use handheld mobile devices in conjunction with the virtual reality hardware. This can extend the interaction possibilities by enabling more users to collaborate together. Websites can be employed to trigger functionalities or configure systems. It can even allow to easily enter text, upload files or act as an additional screen to display images or video. On the whole, web interfaces are very useful for most network communication use-cases. The web server module allows to easily connect to PolyVR, for example with a WebSocket. Many frameworks and systems provide web interfaces and services. An example are PLM systems as described in [GOG07].

5.2.3 Photometric Lightning

Light and shadows are essential components of our visual experience. They provide contrast where surface texture does not. This is even more important in dynamic scenes, where we expect the appearance of objects to change in the play of lights and shadows as seen on fig. 5.5. The standard rendering pipeline is quite limiting when it comes to scene lightning. The hard OpenGL limit of eight lights per triangle is not the main problem, as even three or four greatly impact performance to a non acceptable point, and shadows are really hard to use, depending on the scene requirements.
One state of the art approach to this challenging task is to use a so called deferred shading. The details are described in chapter 2.2.4 of the theoretical background. In short, deferred shading allows to split the rendering process in two parts, the rendering of the scene, with all its geometries, and then the lightning calculation. The lightning is thus done independently of the geome-

try. This means that the geometry is traversed once, independently of the amount of lights in the scene. This allows to place much more lights in the scene as with the fixed rendering pipeline. Shadows are also much easier to integrate.
PolyVR can be switched to deferred rendering with one click. The process behind the scenes is quite complex as the materials have to be exchanged with special deferred shaders. Those shaders are automatically generated based on the configuration of the materials. The underlying deferred shading and shadow system is provided by the OpenSG scene graph library which has been wrapped and integrated in PolyVR.
To further enhance realism, the OpenSG deferred light shading has been extended to support photometric lights. This goes beyond the classic OpenGL light types like point lights, directional or spot lights. They are described in detail in this following section.

The IES standard

The IES [McK47] file format was developed in 1986 by the the Illuminating Engineering Society of North America (IESNA) as the first industrial standard for the exchange of photometric information of luminaires and light sources. The IES format contains the luminous intensities of a light source in all possibles directions. The luminaire describes the intensity distribution in polar coordinates all around the light source. Such data can either be designed with a 3D modelling software or measured from a real light source. This greatly increases the realism of a virtual environment as photometric lights are much closer to the real lights and transport much more details than classic OpenGL lights. PolyVR implements the IES standard. The importer has been validated with dozens of luminaires gathered on the internet, by comparing the import results with available reference images. Photometric lights are not used as broadly as one would expect, they are not available in most of the major 3D modelling tools. There are dedicated viewers to visualize the luminaires, and plugins for 3D modelling tools like Blender, which often support the IES standard quite poorly.

Rendering Pipeline

To use photometric light in the deferred shading rendering pipeline, the intensity distribution has to be transferred onto the graphics card and applied in the deferred lightning shaders to render the scene (fig. 5.6). In PolyVR, the photometric data is processed and packed in a texture. This texture is then used in an extended point light shader where the intensity, sampled from the photometric data texture, modulates the light intensities in all directions.

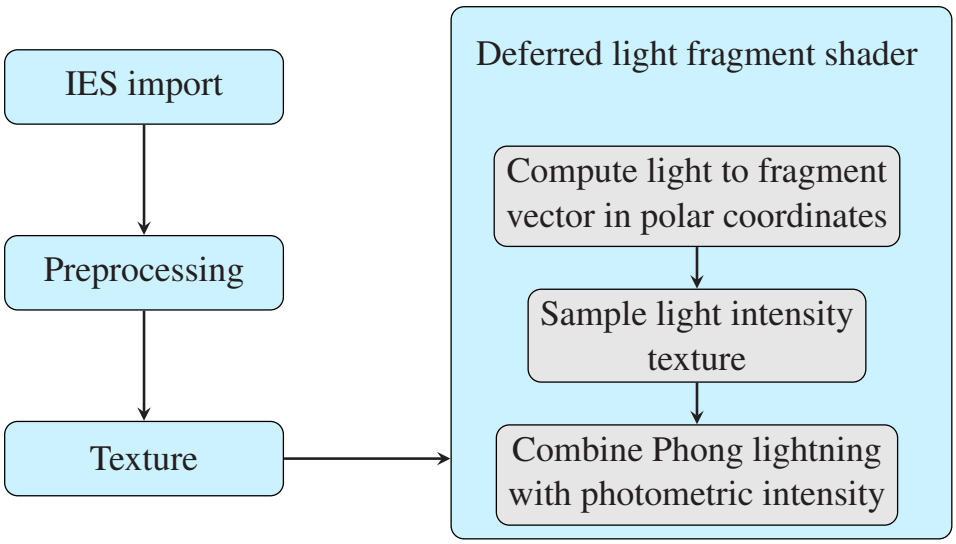


Fig. 5.6: Photometric lights rendering

5.2.4 Scripting Environment

The user interface is the top layer in the software architecture. It contains many widgets which form the integrated development and execution environment. The scripting environment allows the user to develop his application logic using all available functionality of PolyVR.

The scripting environment allows to write scripts to implement application logic, websites, or shaders. The scripting language is Python, the reasons for that choice are that on one hand Python is a very intuitive and flexible language, but on the other hand that it went through a hype resulting in a broad spectrum of available libraries with python bindings. The python interpreter is the Python 2.79 C API. It is a full fledged interpreter that easily wraps the C++ functionality within PolyVR and exposes it to the scripting environment, allowing the user to access all the core modules of PolyVR. Python is quite slow, but this does not matter as it is merely the glue holding together the application logic. It allows to easily instantiate and configure the various core modules, which themselves do the heavy lifting.

The python C API allows to easily write bindings to expose C functions, so called python bindings. To write such a binding requires to add an entry to a static array of callbacks, and to write the callback which wraps the python specifics and calls the C function to be exposed. This callback also parses and converts the python arguments and converts the return value of the C function into a python object and returns it. This results in at least five lines of code just to expose a function that does not take any arguments and does not return anything. For more complex functions, each argument has to be converted, thus requiring even more lines of code. This will do if creating just a few python bindings, but can quickly result in thousands of lines of code when creating bindings for a whole 3D engine. There are alternatives to the python C API for creating python bindings, but they all have some drawbacks.

This is why a python bindings factory was developed for PolyVR. It reduces the code needed to wrap a C/C++ function to a single line. It achieves this by heavy usage of templates, macros, variadic templates and variadic macros. This system is partly used later on for introducing callbacks into the reasoning system 5.5.4.
When creating 3D content, it might be necessary to access the shading pipeline, introducing custom shaders to achieve various goals and effects, not possible in real time using the CPU. To put it simply, using the graphics API allows to pass GLSL code to the graphics card which gets executed there. This can be done at run time, meaning the user can use the scripting environment to write his shaders and directly see the result of each change in the 3D environment. This greatly reduces the development time of such shader code. PolyVR supports the main shaders, fragment and vertex shaders, as well as the more advanced geometry and tessellation shaders. More on shaders is found in chapter 2.2.4. Examples on how to use shader in PolyVR are provided with the Software.
The last usage of the scripting environment is to develop websites. Especially useful for designing 2D Interactive UI in the virtual world. More on this in chapter 5.2.2.

5.3 Virtual Environment Authoring

Creating virtual environments is a broad domain of integrating heterogeneous data and data sources and designing a consistent interactive 3D environment. To narrow it slightly down, the focus as already stated in this thesis is on virtual environments for engineering applications. This section will describe the authoring process of such a virtual environment using PolyVR.

5.3.1 Authoring Pipeline

The authoring process of PolyVR has been designed as seamless as possible. When starting, the PolyVR project instantiates the scene graph and triggers the initialisation scripts. From this point onward the application runs with maximum 60 Hz, depending on the performance of the user's scripts and rendering of his scene. When editing scripts, the changes are directly applied when the script is triggered next. For GLSL scripts, the changes take effect on the next rendered frame, which greatly eases the development of shaders. A change to a website does reload any embedded website in the virtual environment, also making the development easier.
When testing parameter values of components of the scene graph, it is possible to use the scene graph tree view, select components and edit their parameters. Once found, the correct parameter has to be set in the initialization script to take effect for the next start of the application, as the scene graph tree view cannot make the changes persistent for dynamically created components.

5.3.2 Scripting Environment

The scripting environment has been realized using the python interpreter, a script editor widget and a bindings factory to easily expose the functionality of any C++ module. Additionally, the scripting environment allows to write GLSL shader and websites. The choice to use the Python scripting language has been motivated in chapter 2.5.1, as is the use of GLSL and web technologies.
The Python C API provides the infrastructure and data structures to define the PolyVR Python modules and bindings to their C++ implementation. The Python C API is very powerful, but lacks object orientation and requires a lot of redundant code which greatly bloats the code-base. This is fine for smaller projects, but PolyVR has around 140 modules, a count that grows continuously. A Python binding factory has been developed to greatly reduce the amount of code for each module. Each module is reduced to a C++ class with a single member, the structure of Python member functions. Using the C API directly would add a function definition for each binding as well as a type definition of the Python module. This may still be necessary in special cases, where the default type template is not sufficiently flexible.

```
#include "core/scripting/VRPyBase.h"
#include "VRModule.h"

struct VRPyModule : VRPyBaseT<VRModule> {
    static PyMethodDef methods[];
};

newPyType(Module, New_ptr);

PyMethodDef VRPyModule::methods[] = {
    { binding1 },
    { binding2 },
    { binding3 },
    ...
    {NULL}
};
```

The lines with *binding*1, *binding*2, etc. are the wrapped bindings of the member functions of the C++ module. There are two important aspects, the Python type macro *newPyType* and the system to create each binding. The type macro is quite straightforward as it is directly replaced by the definition of the Python type object in the base class.

```
#define newPyType( X, Y, NEWfkt ) \
template<> PyTypeObject VRPyBaseT< X >::type
```

where *X* is the C++ module, *Y* the name of the Python module and *NEW fkt* a function pointer used to instantiate the Python module.
The Python bindings make use of variadic macros and template functions, a multi layer construct depicted on fig. 5.7. The goal is to provide the Python binding implementation without actually writing it out. This means that there are three tasks to solve via templates and macros.

1. Parse the parameter list passed from the user through the invocation of the binding and convert each parameter to the argument type needed for the C++ method.
2. Invoke the C++ method while passing all the arguments, taking into account default values.
3. Convert the C++ return value to a Python object and return it to the user.

The big advantage of this system is that it greatly eases the extension of PolyVR with new modules and the extension of modules with additional Python bindings. This greatly fosters sustainability and reusability of the modules, as well as synergies between the modules.

5.3.3 Content Generation

The virtual scene consists of visual and acoustic assets, usually imported from external resources. But in many cases, it is necessary to change the geometry in dependence of dynamic parameters. This makes modules and algorithms for content generation a key feature for more advanced applications like data visualization, configurators or any kind of highly interactive scenes.

Geometric Primitives

The simplest geometry generators are the geometric primitives. They are created from a set of parameters, like the radius and the curvature resolution to create a triangulated sphere. The 3D primitives available in PolyVR are listed in table 5.1:
Using primitives allows to quickly prototype a virtual scene or test something out. It avoids to use external resources, making it trivial to share the project as the whole scene is generated by scripts, it is thus a stand-alone solution.

```
PyMethodDef VRPyMyModule::methods[] = {
    {"myMethod", PyWrap( myModule, myMethod, "Docs", R, ... ) }
};
```

```
#define PyWrap(X, Y, D, R, ...)                                          \
(PyCFunction) proxyWrap<                                                 \
    VRPy ## X, R (OSG::VR ## X::*)( __VA_ARGS__ ), &OSG::VR ## X::Y,\
    VRCallbackWrapperParams<MACRO_GET_STR( "" )>                         \
    >::exec,                                                             \
METH_VARARGS,                                                            \
PyWrapDoc(Y,D,R,__VA_ARGS__)
```

```
#define PyWrapDoc(F, D, R, ...) \
D " - " #R " " #F "( " FOR_EACH( __VA_ARGS__ ) " )"
```

```
template<typename sT, typename T, T, class O> struct proxyWrap;
template<typename sT, typename T, typename R, typename ...Args,
            R (T::*mf)(Args...), class O>
struct proxyWrap<sT, R (T::*)(Args...), mf, O> {
    static PyObject* exec(sT* self, PyObject* args);
};

template<typename sT, typename T, typename R, typename ...Args,
            R (T::*mf)(Args...), class O>
PyObject* proxyWrap<sT, R (T::*)(Args...), mf, O>::exec(
            sT* self, PyObject* args) {
    vector<PyObject*> params;
    for (int i=0; i<PyTuple_Size(args); i++)
1       params.push_back(PyTuple_GetItem(args, i));

    auto wrap = VRCallbackWrapperT<PyObject*, O, R (T::*)(Args...)>
                ::create();

    wrap->callback = mf;
    PyObject* res = 0;
2   wrap->execute(self->objPtr.get(), params, res);
3   return res;
}
```

Fig. 5.7: Binding wrapping of the PolyVR Python binding factory. Numbers referenced in text.

Plane	sizeX	sizeY	segmentsX	segmentsY		
Box	sizeX	sizeY	sizeZ	segmentsX	segmentsY	segmentsZ
Sphere	radius	iterations				
Cylinder	height	radius	nSides	doBottom	doTop	doSides
Cone	height	radius	nSides	doBottom	doSides	
Torus	innerRadius	outerRadius	nSegments	nRings		
Teapot	iterations	scale				
Arrow	height	width	trunc	hat	thickness	
Gear	width	hole	pitch	nTeeth	teethSize	bevel
Thread	length	radius	pitch	nSegments		

Tab. 5.1: Parametric 3D geometric primitives

Sweep and Constructive Solid Geometry (CSG) Models

There are many powerful algorithms to create geometry, based on parameters or abstract data models. This section will focus on two simple but very versatile methods to create geometry. The first method is called a sweep model. Very common when constructing CAD models, a 2D profile is extruded in the third dimension along a Bézier path. The profile can also be changed along the path depending on various parameters. This can be used in many scenarios, but a particularly effective use is to create 3D representations of assets that are primarily defined by a path. Such an asset can be a road, fence or guardrail for example. This is also very handy for interactive cables, tubes or hoses, as the interaction changes the path of the asset dynamically and the algorithm quickly regenerates the changed segments of the cable.
The second algorithm presented in this section are the CSG models. Those are simply binary tree structures of 3D models or other CSG models. Each non-leaf node is the combination of its two child nodes, using one of three Boolean operations, addition, subtraction and intersection. Leaf nodes are typically geometric primitives, but any closed surface model, a polyhedron, can be used. This is usually the case with sweep models described above, which offers a great synergy between both modules.

High Performance Meta Data

A major advantage of virtual environments is that even though the first goal is to replicate realistic, immersive or at least intuitive environments, nothing restricts the type of information to bring into and visualize in such an environment. A typical example is textual information, for many applications it is useful to annotate elements in the 3D environment, enrich it with meta data. For visualizing text in a 3D environment, the naive approach is to create a corresponding

texture using a font definition and a library that draws the character strings on a bitmap. The problem with text is that if treated as bigger chunks like paragraphs or pages, then two instances are rarely the same. This means that for each chunk of text there has to be new texture. This can quickly impact texture space and performance. The text generation itself can also take a lot of time. On the contrary when breaking it down to a single character, an English text can be described using for example 128 characters if including numbers and some special characters. The key to a high performance visualization of text is to use geometry shaders. The annotation module in PolyVR transforms character strings into points and attaches the text information via ASCII numbers to the vertices of the resulting point cloud. The geometry shader creates the rectangular geometries to display the chunks of the text. Using the attached ASCII numbers the shader computes the UV coordinates of the characters on a generic character texture containing all available characters. As this process is offloaded to the GPU it is possible to visualize tens of thousands of text fragments in the virtual world in real time. This method also allows to move all those fragments around without much impact on the performance.

Molecule Generator

A good example for higher level content generation is a module in PolyVR that generates a molecule based on its formula. A minimal semantic representation is combined with a set of rules to generate the positions in space of each atom of the molecule and the bonds they form.
This module is especially interesting when combined with a visualisation of atoms and bonds using geometry shaders. This allows for huge macro molecules with tens of thousands of atoms in real time. It also allows to animate molecules, for example when visualizing a chemical reaction.
To be able to easily visualize molecules, opens up many new possibilities. Combined with for example a tutoring system, it is possible to use the molecule generator to create the assets used for visualizing tasks and even allow basic interaction with the models. The tasks can be generated themselves based on the knowledge domain to train and the skill profile of the user. The training can be automatically evaluated and new tasks generated to offer a smooth learning curve.

5.3.4 Interaction Modules

A static virtual environment offers very limited added value. Even navigation is a form of interaction. How much interaction a virtual environment offers depends on the complexity of the application logic that is defined by the developer of the environment. This is why it is important to provide a developer with tools and modules for well tested generic interaction paradigms. This greatly reduces the development time and drastically increases the quality of the applica-

tion overall. The more a specific module is used, the better it becomes as bugs and issues are found and fixed.

Navigation

PolyVR offers basic navigation utilities and predefined navigation modes. The first navigation paradigm is the orbit navigation for mouse interaction. The user can rotate around a focus point in space along the vertical and horizontal axis, zoom along the depth axis, and if needed reposition the focus point. This is a very efficient navigation for exploring a 3D environment and is often used in 3D modelling software. Another navigation paradigm is the fly-through navigation using a Flystick. A Flystick is a typical interaction device used in immersive hardware setups and has the advantage of providing its position and orientation in space, in all 6 degrees of freedom. The fly-through paradigm allows us, using a joystick on the Flystick, to fly along the pointing axis of the Flystick and rotate around the vertical axis. This navigation style is not as flexible as an orbit navigation, as it has a maximum movement speed, but it is much closer to natural movement, and thus is very intuitive.

Drag and Drop, Undo Redo

The most basic interaction with objects in a virtual environment is drag and drop. This is one of the first things a child learns, to grab objects and displace them to achieve a goal or to just analyse them close up from all angles. Drag and drop is provided by PolyVR. To enable drag and drop on an object, one has to set its pickable flag to true. PolyVR enables this functionality for most interaction devices by default. The drag and drop operation is implemented internally by rearranging the scene graph. The dragged object is made a child node of the interaction device beacon. On drop, the dragged object is returned to its prior position in the scene graph.
Another module, the SnappingEngine, allows the developer to define rules that change the behaviour of dragged objects. The modules allows to define points, lines or planes in space, global or relative to other objects, where a dragged object can be snapped to. To further extend this functionality, the ConstructionKit module allows to attach snapped objects to each other on a successfully snap event. The module handles the manipulations of the scene graph as well as detaching parts when needed.
A complementary functionality is a built-in undo/redo system. Some methods, if their object is added to an undo/redo manager, record an undo and redo action. The action contains a functor, the reference to the changed object, and the parameter to set if executed. Those actions are stored on a stack internally, calling undo or redo on the manager executes the action on the stack. This synergises well with the drag and drop functionality.

Manipulation Tools

A typical functionality of configuration applications is to manipulate the scene by interacting with geometry. Apart from drag and drop, it is often necessary to offer the possibility of editing visual assets in real time. A basic concept in that regard are handles that visualize the parameters of geometric primitives. Those handles can be interacted with, allowing to change the corresponding parameter continuously, thus changing the geometry. Those handles can be combined with the drag and drop functionality as well as with the undo/redo system.
Another module that is often used is the Pathtool. It is a tool that visualizes Bézier paths or graph structures and optionally adds handles, like the ones used for manipulating geometric primitives. By using sweep models to visualize the paths, the tool can recompute the visuals in real time after each user interaction. This is often used for logistics planning or to interact with abstract graph structures like process visualizations.

5.3.5 Real Time Interfaces

An important functionality for many engineering applications are communication interfaces and protocols, especially for robotics, machine monitoring, training or communicating with web servers. PolyVR does provide the developer with many such interfaces.
The basic network communication follows the REST architectural style. PolyVR uses the Mongoose [Ham10] library to offer full web server functionality, including WebSockets. For more control of systems available on the network, PolyVR uses this SSH [al.04] library to for example configure and manage a cluster for a distributed visualization system.
An industrial communication standard for machinery and production plants and other machinery is OPC UA [LM06]. PolyVR uses the Free OPC UA [al.13] library to provide easy and fast setup of the communication with such systems. A similar system for robotics is ROS, PolyVR does not yet wrap a ROS library, but as ROS offers Python bindings it is nearly as easy to just install them independently and import them in PolyVR through the Python scripting environment.
Yet another typical kind communication use-case is to communicate with hardware devices over the serial bus. PolyVR does provide an implementation for serial communication in that sense, as well as the well known (not so high-level) High-Level Data Link Control [GLP76] (HDLC) protocol. PolyVR even provides AES encryption using the library Crypto++.

5.4 Hardware

This chapter describes PolyVR's extensive support of virtual reality hardware components. They are presented in two categories, the visualization systems and the interaction devices. The emphasis is on how PolyVR enables the user to deploy his content on various types of

systems. The setup, configuration and monitoring are the key aspects from the perspective of the user. We try to enhance and optimize the workflows to make them simpler and safer while retaining the necessary flexibility to support most hardware systems. We complete this chapter with the deployment of virtual reality setups in SMEs, especially for engineering applications.

5.4.1 Visualization Systems

Visualization systems can be composed of one or more displays, connected to a single workstation or connected to different nodes of a visualization cluster. Those displays can usually be monitors or television screens, projection screens or head mounted displays. They have in common that they output a visual feedback to the user, based on the input of 2D image data. This section focuses on the configuration of such systems with PolyVR.

Windows and Views

Any display as seen from PolyVR is basically a 2D image, placed in a 3D environment. The exact position of the display surface in space allows, in combination with the head position of the user, to simulate a window, from the real world into the virtual world. The head position can be detected using state of the art tracking systems. The important aspect regarding the display configuration are the display coordinates, size and pose, in the tracking coordinate system. This coordinate system is defined when calibrating the tracking system used for capturing the position of the user for head and hand tracking.
The view geometry is defined by its width and height in meter, as well as a shear and a warp factor. The view pose is defined by a position vector, the center of the view, the normal vector of the view and the up vector of the view. This fully defines a view in the tracking space. This allows to transform the display to behave like a window, a window from the real world into the virtual environment. To achieve this effect the rendering of the virtual environment has to use one to one scale and adapt continuously and in real time to the perspective of the user. The virtual scene has to be rendered based on what he sees from his position, through the frustum spanned by corners and sides of the display.

Cluster Environments

A display requires image data that has to be provided by a graphic card. Depending on the available hardware and the targeted display resolution, a single graphic card can provide for even three or four displays. It is even possible to put multiple cards in one workstation. This allows to power complex visualization systems using a single node, avoiding any clustering overhead. One example could be using three graphic cards with four display ports allowing for up to twelve full HD displays. This could be used for a power-wall of four times three televisions, or a CAVE system with six projection screens using passive stereo, which would

also require a total of twelve images. Such a system would be very reactive. All synchronization of the rendering would happen locally, but the load for each graphic card would be huge. The total amount of triangles that can be rendered at sixty Hertz would be roughly divided by four due to the splitting of the hardware resource between the displays. When the usage of the system encompasses cases with huge amounts of data, for instance scientific data visualization or point clouds from 3D scans, then the system should be extended with multiple nodes and more graphic cards. One example would be the six sided projection system from above, one graphic card for each side might be a good idea, where each card renders a side by side image, two times full HD. The six cards could be distributed on three workstations. This would ensure that the load on the CPUs is not too high and the overhead of the clustering is also not too big.

5.4.2 Interaction Devices

There are various hardware systems to allow interacting with virtual worlds. Most offer tracking of 3D bodies with all 6 DoF, and signals like buttons and sliders. More exotic systems like force feedback devices or gloves have their niches as well. A typical controller example is the Flystick from ART, it has 6 DoF, a few buttons and a joystick. It allows most basic types of interaction, while being quite precise, ergonomic and intuitive. The virtual reality software provides the configuration of the interaction devices, examples or navigation and interaction paradigms, and the necessary API to script custom, application dependent, interaction logic. The first goal of PolyVR is to ease the configuration of new devices. The ART protocol sends UDP packages on a specific port. It is enough to specify that port in PolyVR, all devices get automatically detected and added to the system. The VRPN protocol requires to specify the device address, the VR software takes the role of the client and receives the device signals. A device always owns a beacon that defines its pose in space. The windowing or tracking system continuously sets the position and orientation of the beacon. It is possible to combine the tracking from one system with the device of another. This is useful for example when developing a new interaction device using the VRPN protocol to send all device signals, but combining it with a tracked body of an available ART system.

A special type of interaction device detects the position and configuration from the hands or the body of the user, without any additional hand-held or attached hardware. This is a very complex topic as described in [HHHO14]. It is possible to use Kinect or Leap Motion devices. They track body or hand configuration input to achieve such interaction, but the currently available sensors and algorithms are very limited in accuracy and usability. The main issue is the support for pointing and precise motion necessary for drag and drop for example. The human body, and especially his hands, are complex kinematic systems. Their geometry varies a lot between individuals and changes continuously when interacting.

PolyVR tries to partly unify the data model of interaction devices to further ease the development of immersive interaction logic. Instead of the usual 2D coordinates, the user has to handle mouse interaction in 3D, as for all other devices. The mouse device owns a beacon like any other device. This beacon is at the same position of the active camera, thus in the middle of the viewport. The position does not change when moving the mouse, only its orientation. The direction is computed internally from the 2D viewport coordinates of the mouse. This approach allows to write the interaction scripts once for most types of devices. The user is not burdened with translating 2D to 3D paradigms and vice versa. This also eases the deployment of a virtual reality application to an immersive hardware setup.
Force feedback devices need a physics engine to allow the user to interact with virtual objects. The device usually controls directly the pose of a virtual object or avatar. When collisions occur, forces are generated from the physics engine. The device has to transmit those forces to the user to achieve force feedback.
Another common device are driving simulators. They are build for the specific use-case of driving virtual cars or other vehicles. Other devices like flight or boat simulators have similar characteristics. Commercial gaming equipment can be easily added to the immersive setup via libraries like Pygame. More advanced setups using real car components may be integrated using a CAN-bus interface. This requires reverse engineering of each component. PolyVR comes with various interaction and navigation paradigms.

5.4.3 Virtual Reality Systems in SMEs

The hardware is an important hurdle for SMEs when thinking about investing in a virtual reality solution. The supporting role of VR in the engineering workflows makes it difficult to estimate the potential added value of such investments. This is a reason why especially SMEs have been reluctant to adopt VR technologies. This is currently changing fast, there are a number of factors that make such technologies recently more attractive. One factor is the hardware. The technology has become better and better over the years, requiring less maintenance and the costs have drastically dropped. Especially the services regarding the installation and configuration of advanced visualization setups have become more efficient and thus more cost effective. It is still difficult for most companies to decide an investment in virtual reality technologies. Jana Dücker has developed a methodology to evaluate the benefits of such technologies taking the specifics of the company into account [DHO15, DHO16b, DHO16a, HDO15].
PolyVR does try to maximise the added value for a broad spectrum of use cases, but the more advanced features and optimizations have definitely a focus on VR for engineering applications, especially for production plant engineering. A typical example is the 3 sided projection system with a front, side and floor projection surface. This offers a great field of view as well as enough space to enter the collaborative environment with two or three persons. Tracking systems are

costly, but they are imperative to use multi-sided CAVE environments. They track advanced interaction devices and the position of the user, which is used to render the virtual environment as a consistent space from the point of view of the user. An additional display can enhance the system by displaying the undistorted view of the person that is currently tracked by the system. This helps persons standing next to the system to see and understand what the user is focusing on.

5.5 Virtual Engineering

The next major industrial revolution is coming in the form of major paradigm shifts in product development. The pressure that drives those changes is the result of globalization and the steady advances in IT technologies. Globalization of markets, product and information flows forces the companies to innovate at a steadily accelerating pace, and the exponentially growing capabilities of electronics and software innovation greatly reduce the duration of technology life cycles, also for established industries like machinery and plant construction. There lies a gigantic potential in closing the gaps between disciplines. This is an ongoing trend. An example is the emergence of the mechatronics course of studies, combining mechanics and electronics courses.
The traditional separation between mechanics, electronics, software and automation is a growing issue for companies. This translates into different departments inside of the companies, employing engineers with different formation and domain, using different CAD software. But those departments have to work together on the same product, and splitting the product development by mechanical, electronic and software components is increasingly difficult.
This is why a lot of effort is put into bringing the domains together, especially on data and software level. This means that the various data produced in different CAD software tools has to be integrated in a single holistic data model of the product, the so called virtual twin.
This section describes the methodology used to build up the virtual twin, especially the MCAD and ECAD interfaces, the automation virtualization, and the semantic layer used to integrate all the data.

5.5.1 CAD Data Exchange

An important basis for the next industrial revolution is the widespread use of CAD systems. Nowadays one can say that this is mostly given. The greatest hindrance to the further automation of the virtual engineering validation iterations are the CAD interfaces. There are various formats for the exchange of CAD models. Those formats can be categorized in two kinds: meshed surfaces and BREP models, further described in chapter 3.4.1. Historically, virtual reality frameworks mostly work with meshed surface models. Those are easy to handle and visualize as they basically reduce complex models to sets of simple geometric primitives like lines, triangles and quads. Graphic cards are designed to process those primitives in a highly efficient

way. The downside of those models is a huge loss of not only geometric precision but more importantly meta data. This drastically limits the possibilities to integrate semantic information in the virtual environment.
Any virtual environment that consists of more than static geometries is usually manually designed in all aspects. CAD models are refactored and textured using 3D editing tools, dynamic parts are animated and interactive aspects are scripted. This is manual labor that can take months or even years, depending on the complexity of the scene.
The alternative exchange formats based on BREP models are very promising as they can address the data loss issues described above. BREP formats have the following advantages:

- BREP models have much smaller file sizes than tessellated models
- Parsing data files requires less memory overhead
- more information is available like the scene graph, meta data, materials and dynamics and much more
- CAD tools and the VR environment work with the same mathematical description of the scene geometries. This allows for instance to synchronise changes between the systems.
- When using BREP, one has the control over the tessellation and can thus optimize it or automate the generation of level of detail models

There are multiple formats that can export BREP models like IGES, STEP and JT. As motivated in section 3.5, this work focuses on the STEP AP214 format. The Open CASCADE library is used to parse the STEP files. The imported data structure is traversed to build the scene. Each part is tessellated using the Open CASCADE tessellation module and its material data is gathered and attached. The scene graph of the model is reconstructed following the assembly structure.

5.5.2 Other Data Interfaces

PolyVR does support many geometry formats, most using the import modules of the OpenSG scene graph library. All formats are listed in table 5.2. The formats labeled under PolyVR are fully implemented in PolyVR. The OpenSG importer are barely wrapped as they also construct the correct data structures. Other importers are moderately wrapped as the results have to to be post processed into the scene graph data structures.

5.5.3 CAD-VR Interface

To bridge the gap in the data workflow between CAD software and the VR software PolyVR, a choice was made to develop a plugin with the aim of gaining full access to the CAD data

Format	Type	Library
VRML	Mesh	PolyVR
OBJ	Mesh	OpenSG
STL	Mesh	OpenSG
COLLADA	Mesh	OpenSG
PLY	Mesh	PolyVR
VTK	Mesh	VTK
SHP	Mesh	GDAL
PDF	Topography	GDAL
TIFF	Topography	GDAL
HGT	Topography	GDAL
IFC	CAD	Open CASCADE
STEP	CAD	Open CASCADE
E57	Point cloud	E57Foundation
XYZ	Point cloud	PolyVR

Tab. 5.2: Geometry exchange formats supported in PolyVR

model and transmitting it to VR. In addition to interfacing the CAD data, the plugin allows for a bidirectional interface. This opens up many opportunities of interaction between the CAD and VR software. The plugin has been implemented for two different CAD software tools, Siemens NX and Dassault SolidWorks. Further implementations are planned, especially for ECAD software like ePLAN.

Various features have been implemented to support design reviews in VR during the product development process. The primary objective is to greatly simplify the virtualisation workflow by directly sending the CAD data to VR without using an exchange format like VRML. This does greatly reduce the amount of time needed to transfer the construction data. The second objective is to further reduce the time for iterating by directly implementing the results of the design review on the fly in the CAD model. This is done at a CAD workplace directly adjacent to the immersive CAVE environment. Instead of recording the necessary changes in a protocol and scheduling a new design review the next day, the changes are applied directly in the CAD software. Those changes are then transmitted automatically through the CAD plugin. This has the advantage that the model does not have to be retransmitted as a whole, just the changes like remeshing a part, synchronizing a transformation, or updating the part material. To further improve this workflow, the communication between the users in the CAVE environment and

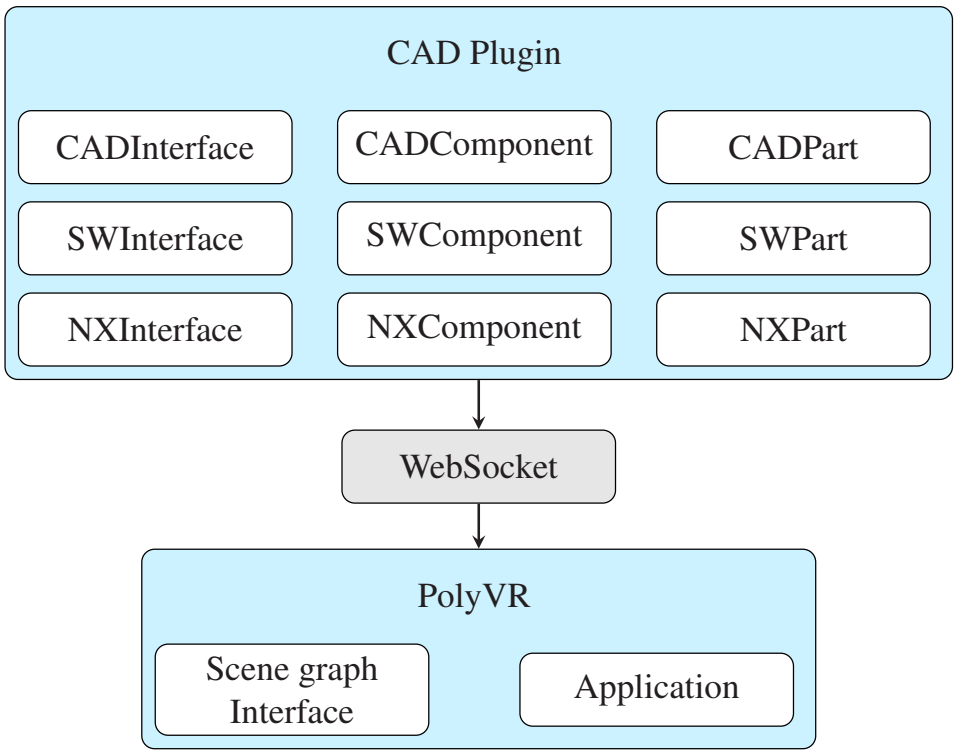


Fig. 5.8: Software architecture of the CAD plugin for CAD-VR data exchange

the construction engineer is supported by transmitting the part selected in the CAD software, highlighted in VR, or the geometry selected in VR highlighted in the CAD tool.
The design of the plugin is based on two premises. The first premise is that the plugin should be modular and extensible. The first module is the assembly, the scene graph structure and geometries, and the second module are the kinematic and dynamic information. Other modules will be developed accordingly. The second premise is to be easily transferred between CAD software version changes or between different CAD software manufacturers. The software architecture of the plugin is depicted on fig. 5.8. The primary data model of the plugin is based on the part and assembly component paradigm of most CAD software. This means that a part represents a geometry and a component is a node in the assembly scene graph that represents an instance of a part or a structural assembly node. An important difference between parts and components, is that parts consist of geometric data without hierarchical structure. Most of the metadata like visibility, materials, transformations or kinematics are defined on a component basis. A component is a node in the assembly structure that also references a part. It allows to reuse the same geometric data in different places of the assembly but at the same time allows to apply different materials and even additional features to the individual component. This reduces the amount of modelling work as well as the computational resources to handle and display the assembly.
The basic architecture of the plugin software emphasizes the minimal cohesion with each individual CAD software APIs. It is important to wrap the APIs and abstract as much of the plugin functionality into API agnostic modules. Those are the three classes, CADPart, CADCompo-

nent and CADInterface. They define the functionality, workflows and communication of the CAD plugin. To put this into perspective, the first implementation of the plugin for SolidWorks took six months of development. The second implementation for Siemens NX only took less than a week. The API dependent functions needed are as follows:

- Traverse the CAD scene graph, extract parts and components.
- Parameterize and execute the tessellation for the parts.
- Access components visibility, material and transformation.
- Access signals to attach callbacks to parts, components or the main CAD interface.

Now follow some implementation details. The plugin is realized as a C# library, loaded as an extension when starting up the CAD tool. The bidirectional communication interface is a WebSocket, used to transmit the initial CAD model, and then the changes to the data model or events due to user interaction. The plugin has a set of configuration parameters, stored in the registry and editable through a settings dialog in the CAD tool, as depicted in figure 6.26. The VR host IP and port configure the WebSocket connection. The other parameters configure the tessellation quality. This allows to optimize the performance for big CAD models.
To summarize and conclude this section, the CAD plugin that has been developed allows to drastically shorten the CAD validation iterations during the construction phase of the product development. It greatly simplifies the workflow to bring the CAD data into VR, and also allows for bidirectional communication between CAD and VR. This offers an optimal collaboration ground to discuss and directly apply changes to the CAD model. It is also the basis for further integration of ECAD data as well as automation data, towards the fully functional virtual twin. The next section will address the mapping of the geometry with generic semantic knowledge, as an attempt to automatically infuse the CAD data with intelligent behaviour.

5.5.4 Semantic Layer

The classical approach to add dynamic and interactive behaviour to a virtual scene is by scripting everything (fig. 5.9). This results in a rigid user experience, it is limited to the amount of resources and effort put into the programming. The semantic layer is on the other hand a system that allows to add intelligence to the virtual environment (fig. 5.10). Other aims of the system are to ease the use and reuse of application logic elements and interaction paradigms and to further automate the authoring process as much as possible.

Knowledge Base

The data model of the semantic layer is also known as a knowledge base, the theoretical background is explained in chapter 2.4. As described, such a knowledge base consists of an Ontolo-

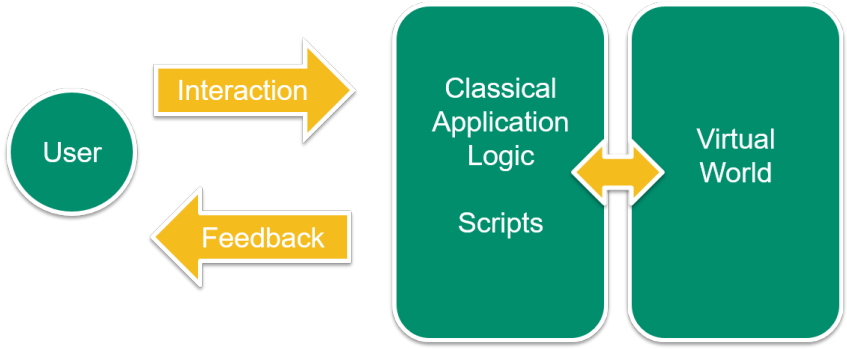


Fig. 5.9: Typical interaction loop. The typical approach to script the application logic is slow and requires expert knowledge.

gy, a taxonomy and rules, and a set of entities. This chapter will show how those technologies can be used in virtual environment, especially for authoring of engineering applications. The first task when creating a virtual environment is to define the content. Most visual content like 3D models usually lack semantic description. The data does not have a classification that could allow further processing. The virtual reality system needs such a classification for further automating the authoring process. One goal is thus to add that classification information to reduce the work needed for adding interaction and application logic. On one hand, this can drastically reduce the amount of scripting necessary for every new virtual scene. On the other hand it can also greatly ease the description of complex behaviour and even instantiate and automatically parameterize simulations like car dynamics or kinematic systems. Especially basic interactions like opening doors, clipping the navigation to the floor or toggling light switches can be enabled in a scene without the need to explicitly script those behaviour. The goal is to get the system to a point where it is capable to fully automate the creation of a fully functional virtual model of complex machinery like production plants. This would allow to greatly reduce the work and time for many applications like virtual commissioning, maintenance simulations or training machine operation.

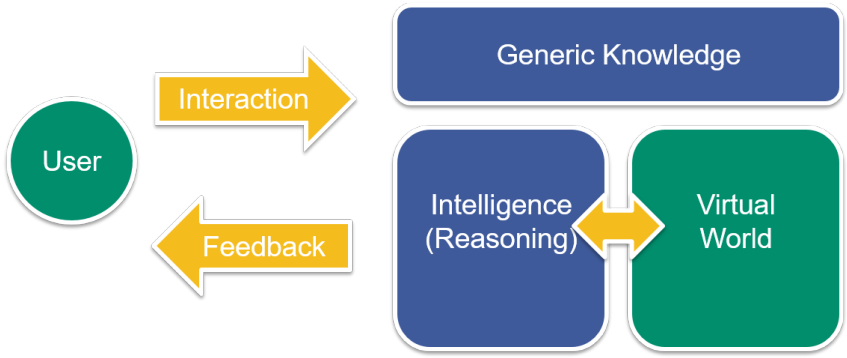


Fig. 5.10: Using generic knowledge to infuse the virtual world with intelligent behaviour and reducing the need to script a complex application logic.

Reasoning

The active part of the semantic layer is the reasoning module, the theoretical background of which is explained in chapter 2.4. As described, the reasoning system allows to process queries to the knowledge base. It has the ability to infer information and make assumptions. In this chapter, we show how to use it for intelligent interactive virtual environments, where the data to analyse is essentially the semantic information, the knowledge of the virtual scene and where the queries are usually triggered through interaction of the user with the scene or by algorithms. The basic idea on one hand is to reduce the amount of work in order to create complex interactive applications as mentioned above, but on the other hand to go much farther than it is possible with the current means of scripting application logic by hand.

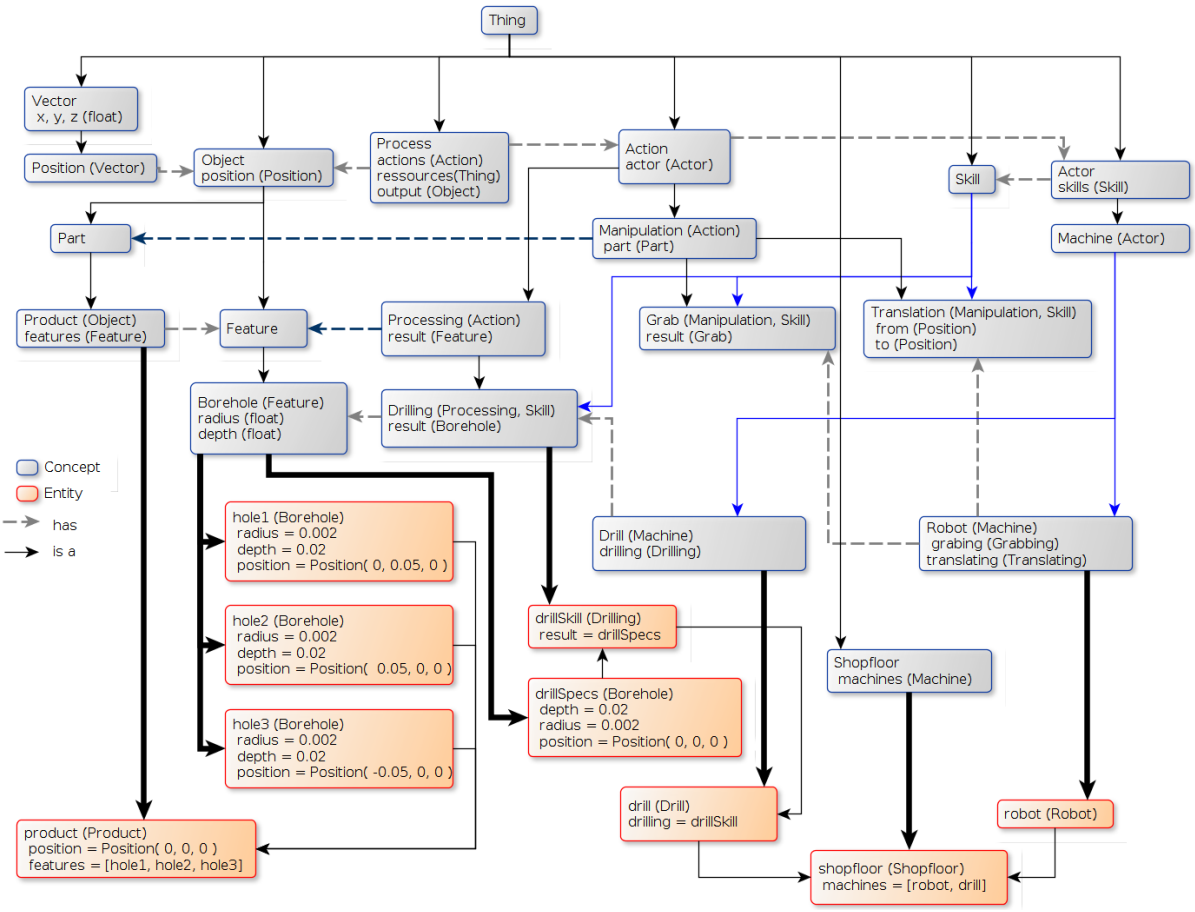


Fig. 5.11: Example Ontology, Process Planning Example

The reasoning process is illustrated in figures 5.11 and 5.12. The first image shows the taxonomy, concepts with properties, and instantiated entities for a minimalist process planning example. The scenario is as follows: a product is defined by a list of features, in this case three boreholes, and this product is supposed to be produced in a shop floor with two machines, a drill and a robot arm. Each machine has skills that allows it to execute actions. Such a concept has been used for example in the works of Alexandrov et al. [ASO14]. The skills of the drill

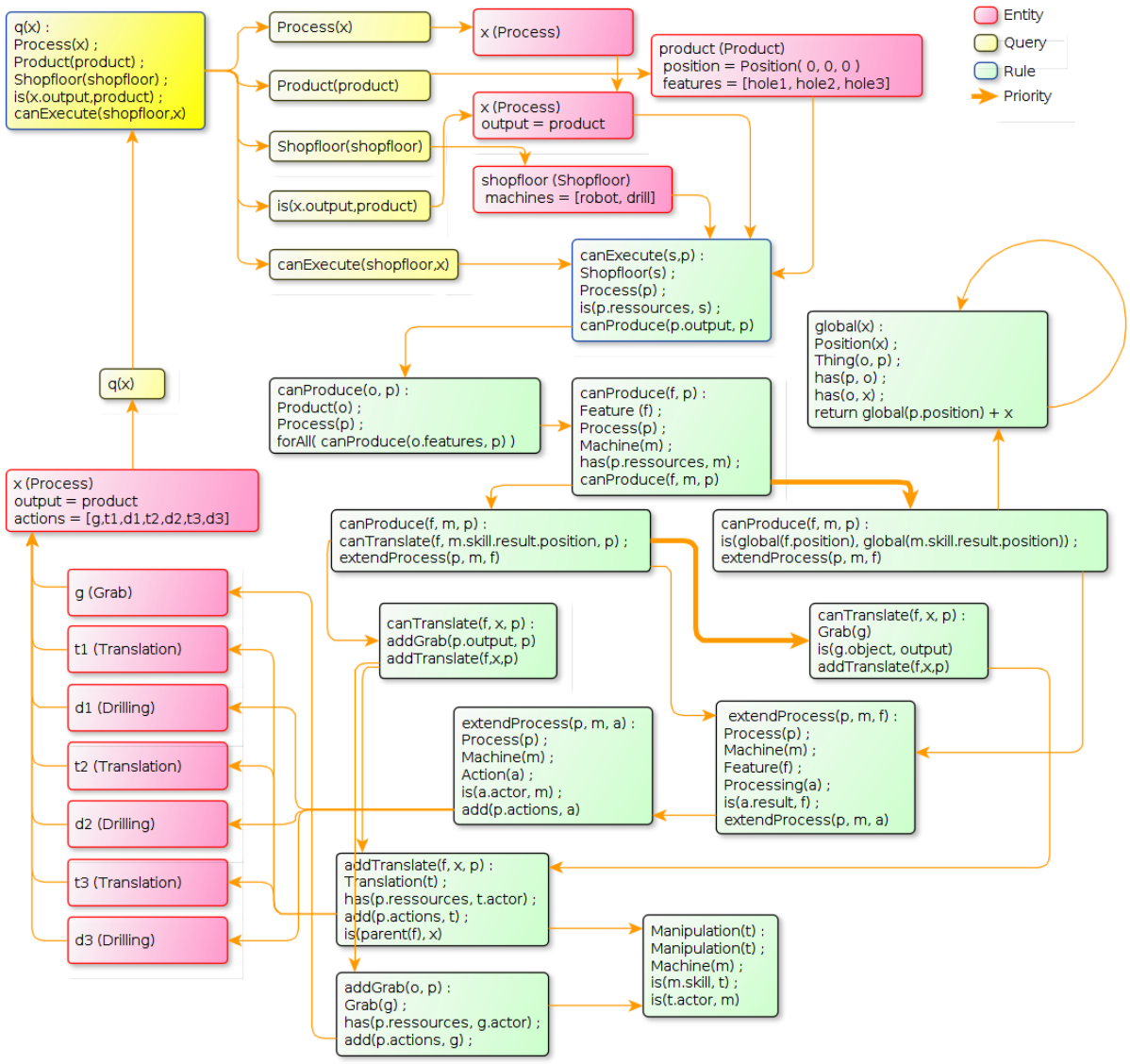


Fig. 5.12: Example Reasoning based on Rules

and the robot are also actions, they inherit from both concepts, skill and action. Multiple inheritance allows in some cases to reduce the complexity of a taxonomy but also imposes the use of a graph structure instead of a simpler tree structure. The taxonomy has been built with the following guidelines in mind:

- Use the English language, the meaning of a concept should be its most common definition.
- Choose concept name that are short and intuitive.
- Avoid combined words, and no numbers or special characters
- Prioritise structures that are easily extendable and reusable

- Find a balance between the width and the depth of the taxonomy.
- Keep it simple.

A good taxonomy should be intuitive and the relations should be easy to explore and grasp. An advantage of those guidelines should be to facilitate the merging and mapping of different ontologies.
The second figure shows a visualization of the reasoning process, at least on a certain abstraction level. The figure depicts a graph that links the starting point of the reasoning process, the user query, to the entities and rules involved in the process and to the generated result returned by the query. The reasoning process has been depicted in a very simplified manner to visualize the basic flow of information from start to end. The rules are generic rules defined in the ontology, as well as the entities for the shop floor and its machines as well as the product and its features. The process is an entity instantiated by a statement of the query, and the process actions are instantiated throughout the reasoning process. The process starts with parsing the query, splits it into statements and evaluate them individually. This is also the same way rules are processed when added to the local reasoning context. The second step is to analyse each statement, this can be a type definition for a variable, resulting in the instancing of a variable pointing to a known entity, a set of entities or a temporary anonymous entity. It can also be a built-in method like the basic verbs 'is' or 'has' or a reference to a rule.
The implementation of the reasoning system in PolyVR does contain a parsing algorithm that transforms the strings into chunks and structures them in a tree structure. This functionality even handles math tokens like brackets, operators or vectors and can compute the results of those mathematical expressions. The next important module is the handling of the reasoning context, variables, substitutions in rules and much more.
The reasoning module is an important stepping stone towards the automated virtual commissioning. The virtualization workflow as presented in chapter 2.7 uses the reasoning system in two steps of the virtualization process, once for semantically enriching CAD data to integrate MCAD and ECAD data, and then to attach the functional behaviour to the CAD components. The virtualization topic is further addressed in the validation chapter 6.2.3. Overall it is important to say that the reasoning system in PolyVR is still very rudimentary and will need much further development to reach a maturity level that allows to use it productively for complex virtual environments. Especially regarding performance it is critical to find strategies to make the reasoning much more efficient.

5.5.5 Kinematics

The chapter 4.3.7 describes the method to recreate the dynamic behaviour of mechanical systems. The goal is to simulate kinematics and mechanisms of CAD machinery and allow the

user to interact with the HMI components like levers and wheels in the virtual environment. The process starts with the static CAD parts classified by type like gears, axis, chains, etc. .

Geometric Analysis

The first task is to implement the geometric analysis of the parts to extract the relevant parameters for the simulation. The analysis for gears for example uses multiple steps to compute the various parameters.

Gear parameters:

- Rotation axis
- Hole radius
- Gear radius
- Pitch
- Teeth size

The parameters are computed based on the gear geometry. The steps and algorithms to compute the parameters are as follow:

Gear segmentation:

- PCA - compute the direction of the **rotation axis**
- Project the vertices into the rotation plane
- Compute the polar coordinates of the projected vertices

  Minimum radius corresponds to the **hole radius**

  Maximum radius corresponds to the **gear radius**
- Apply a Fourier transformation in polar coordinates

  Compute the main frequency
- Fit a sine curve to the points using the computed frequency

  Invert the frequency to get the radial pitch

  Multiply the radial pitch with the gear radius to obtain the **pitch**

  Sine amplitude corresponds to **teeth size**

The primary component analysis (PCA) has been implemented according to a common algorithm. First the covariance matrix is computed using the geometry vertices. Then the eigenvectors and eigenvalues of the covariance matrix are computed. Comparing the differences of the

eigenvalues will indicated which of the eigenvectors is different of the other two. This eigenvector is the rotation axis of the gear.
Once the rotation axis is known, it is possible to project the vertices into the rotation plane of the gear and compute the polar coordinates of the vertices. By iterating through the vertices the minimum and maximum radii correspond to the hole radius and gear radius, the first two parameters of the gear. Then the polar coordinates of the vertices can be further analysed by deriving a function and doing a functional analysis. To assemble this discrete function the coordinates have to be sorted according to their angular component. Then the point set is re sampled to obtain a discrete function with equidistant points. The gear teeth are a periodic feature that can be analysed by using the fit of a sine function. To estimate the sine function one first needs the frequency of the periodic feature. Using a discrete Fourier transformation the main frequency can be computed. Once the frequency is computed a least square method can be used to fit a sine curve. The sine curve yields an amplitude, frequency and vertical offset, those are the numbers used to compute the final parameters of the gear. The amplitude corresponds to the size of the gear teethes, the vertical offset corresponds to the mid teeth radius of the gear and the inverse of the frequency corresponds to the angular pitch. The angular pitch, multiplied by the mid teeth radius of the gear, yields the gear pitch.

Simulation

The behaviour of the kinematics and mechanisms is implemented using two different simulation modules. The first simulation handles kinematic systems with constraints and joints. The implementation has been realized by using the Bullet physics engine instead of a custom kinematics simulation. The physics engine does provide the necessary features like handling 6 DoF joints and constraints. One difficulty to use the physics engine and physicalize the CAD parts is that the resulting behaviour is too dynamic. The parts are moved by gravity and collisions with other physics parts. There is a possibility to avoid this behaviour and instead obtain a more functional behaviour needed to represent machinery. The key is to properly parameterize the physicalized parts:

- physicalize parts as dynamic rigid bodies
- set linear and angular damping to 1.0
- set gravity to (0.0, 0.0, 0.0)
- set collision margin to 0.0
- set activation mode to 4 which disables the automated deactivation
- set collision group to 1 and and mask to 0 to avoid self collision

The second simulation is handling the components found in more complex mechanisms, gears, screw threads and chains. A custom simulation has been implemented following the method described in chapter 4.3.7. Engines, gearing, robots, production plants and many more contain of more or less elaborated mechanisms. They are the dynamic components giving life to mechanical devices. To be able to correctly simulate them is an important step towards fully interactive scenes.
The mechanism simulation proposed in this work is based on the following requirements:

- The simulation is able to simulate big systems without numerical loss of precision over long propagation chains
- The simulation interfaces seamlessly with other modules like the physics engine or user interaction
- The simulation runs in real time

The major design choices were to favor a purely analytical simulation over a numerical simulation. This means that the simulation works with a more abstract data model based on the parameter of its mechanical components and not on the simulation of colliding gear teethes.
An important aspect is how to combine both simulations. On one side the mechanism simulation runs with one tick per frame and on the other side the kinematics simulation runs with 500 Hz in the physics thread. The problem is that both simulations control the same parts. The physics engine tries to apply the simulated transformation each frame to the geometry transformation. The mechanism simulation reacts to an exterior transformation change, but then also propagates the change through the system and changes the transformations of its components. To avoid interference and dynamic artifacts between the simulations it is necessary, before allowing the mechanism simulation to change a transformation, to switch the physicalized objects from dynamic to static and switch it back to static on the next mechanism iteration.

5.5.6 Wiring and PLCs

The ECAD data and PLC programming is exported from their corresponding modelling tool as XML files. The data this implementation is based on comes from EPLAN and TIA Portal, German editions. The ECAD data is split in several files:

- **Machine.epj**, main project file
- **bmk.edc**, ECAD components
- **connections.edc**, ECAD connections, the wiring graph

The XML files are parsed using a third party library. The BMK file contain a list with ECAD components, mainly the ECAD ID and the component name are extracted from the file. The components are stored in XML nodes with the O17 tag and the parameters are stored as node attributes. The ECAD ID attribute is named P20006 and the component name is stored as attribute P20100_1. The next file, connections.edc, contains the wiring graph. Each connection is described in an XML node with tag O18. The node attributes P31011, P31019 and P31020 contain respectively the wire label, the source address and the target address. The addresses follow a specific scheme that has to be parsed. The address is build as follows:
machine - ***component*** - ***socket*** : ***port***
The source and target address strings have to be parsed according to the scheme above. Some difficulty comes with variations of this scheme, special cases that have to be handled. The ECAD IDs are used to combine the connections with the components. The ECAD ID is simply derived from the address as ***machine*** - ***component*** The socket and port of each address are relevant for the simulation of the wiring. The last file, the main ECAD project file, contains additional information, especially all the ports of a component and the mapping of some ports to LAD variables.
The PLC programming is also exported as an XML files. The data is split over many files, based on an overarching structure of the machine modules. The data used during the implementation of the LAD emulation is split in the following files:

- **Process.xml**, contains the program logic
- **Default_tag_table.xml**, contains variables with hardware addresses
- **Process_Data.xml**, contains variables used in the main process
- **HMI_Data.xml**, contains variables related to HMI programming
- **Alarms_Data.xml**, contains variables used for the alarms system
- **VFD_PAW.xml** and **VFD_PEW.xml**, contain variables for the engine control module

The main file is the Process.xml which contains the programming of the PLC. The other files contain relevant variables necessary to emulate the programming and interface to the wiring simulation. The content of the main file is structured as so called compile units that each contain a fragment of the whole programming. Each compile unit contains a number of operators or blocks that have in and outputs. Those operators are connected to variables that are evaluated when the emulation reaches this point in the program.

Simulation

The wiring and the PLCs are simulated using graph traversal methods. A function starts at some point of the graph, evaluates the initial node and is then propagated to other nodes, depending on the evaluation of the node and the traversal strategy.
For the wiring, the simulation essentially follows the electric current. The data model, graph nodes and edges, are the electric components as nodes and the wires as edges. The propagation strategy is to define a stack with initially the main power supply node. From there, until the stack has been fully processed, the top most node is taken from the stack, evaluated, and depending on the result the connected nodes are put on the stack. Every node that has been evaluated is flagged to avoid evaluating them twice. The components have different ports, when current is on a specific port the simulation has to compute what other ports the current is propagated to. Only a few types of components have been implemented. The most basic component like a fuse simply lets current through. A more complex component is the switch where its state define if the current can run through or not. The most complex components are the PLC modules, where their programming defines their behaviour and thus the current on the output ports.
The LAD emulation system is similar to the wiring simulation. The first difference is that the stack is not constructed and processed continuously, but it is fully constructed with all nodes to be evaluated before the actual evaluation starts. In addition the stack does not only contain the nodes but also the wires to ease the traversal. The stack is constructed by traversing the graph, starting with the so called power rail node. When traversing the graph the traversal of the parallel nodes is prioritized over the next nodes as explained in chapter 4.3.6. To finish the setup of the stack, the redundant nodes have to be removed. Once setup, the stack is used to evaluate all nodes, LAD operators and programming blocks.

5.5.7 Virtual Engineering Implementation Summary

This chapter explained how the virtualization method described in 4.3 has been implemented with PolyVR. The data model can be used to bridge the mechanical, electronic and software domains in the product development process. Semantic web technologies allow to integrate all MCAD, ECAD and automation data into a single data model, the so called virtual twin. Interfaces have been developed and workflows have been defined to integrate the heterogeneous CAD data and create a consistent semantic layer. The semantic layer is automatically built by mapping the geometric parts to ontology concepts. This allows to automatically enrich the scene with intelligent behaviour. The result is an interactive VR model, including dynamics and kinematics, as well as functional logic. The kinematic system, mechanisms and kinematic chains, are simulated by combining two separate simulation systems. An analytical simulation for gears and chains and a simulation based on the Bullet physics engine that handles kinematic chains, joints and constraints. The simulation modules are parameterized using geometric analysis me-

thods to compute all the relevant parameters for each component of the simulation model. The wiring comes from the ECAD data. It is simulated to compute the behaviour of the machinery, the way PLC modules control actuators and process user and sensor input. The last simulation module is the emulation of the PLC programming to fully achieve the virtualization with the functional behaviour. The result is a system that allows to create, in an automated manner, a functional virtual model solely based on the CAD construction data. In the chapter 6, various examples show the capabilities of the current system and demonstrate the validity of the methodology and concepts.

5.6 Implementation Summary

PolyVR started as a validation implementation for experimenting with VR technologies and showed their usefulness for various engineering applications, especially for machine engineering, product development, production planning, training and planning applications. It has since grown into a full fledged product, an authoring environment for VR applications, packed with interfaces to various systems and exchange formats, from 3D mesh data like VRML and COLLADA to BIM and GIS formats like IFC, over CAD formats like STEP and DXF, semantic web data as OWL RDF, interfaces for automation using OPC UA, robotics using ROS, or REST communication using the integrated web server. An intuitive but powerful scripting environment allows to dynamically program virtual environments without stopping and restarting the application. This allows a very efficient authoring workflow.
The broad array of modules allow to quickly implement complex application logic. The user has access to all aspects of the scene graph, physics engine, math modules and much more through the scripting API. The state machine, process engine and semantic reasoning modules allow to setup complex intelligent behaviour in a virtual world. The synergistic nature of the modules allow to easily mix them together, which makes developing with PolyVR very efficient.

6 Validation

This chapter presents two use cases to demonstrate the capabilities of the PolyVR virtual reality authoring system. The aim is to validate the system design in regard to the research questions identified in the introduction chapter. The emphasis is laid on the possibilities offered by the VR framework to support the virtualization process for engineering applications.
The first use case is the generation of large scale open world scenes, necessary for many applications. The specific example will be a driving simulator, using GIS data to create a virtual representation of real world road networks and urban environments as well as a tutoring system for learner drivers.
The second use case is the virtualization of integrated production lines to support the plant engineering product development process. This is a complex domain, connecting to most CAD domains like MCAD, ECAD, BIM and automation. It can also be extended to interfacing PDM and ERP systems to further integrate in the product development process of the SME.
This relates back to the research questions in this thesis that are basically threefold. The use-case examples in this validation chapter will show the innovation of the VR system as a new authoring environment for virtual environments, its advantages for creating engineering applications, as well as the benefits for SMEs to use PolyVR to deploy new virtual engineering workflows.

6.1 Driving Simulation

Driving simulators are typical examples of virtual environments. They are usually characterized by a mixed reality setup, a driving simulation and a virtual world with roads to drive on. The basics of open world environments are explained in chapter 2.6. This chapter will describe the project of a driving simulator developed with PolyVR. This can be considered as a use case to validate the following aspects of PolyVR:

1. Validate the capabilities of PolyVR for content generation, especially for open worlds.
2. Validate PolyVR interfaces for GIS data.
3. Validate the use of reasoning for interactive applications, especially as a basis for a training application.

The following sections describe the driving simulator developed at IMI based on PolyVR. The figure 6.1 shows an overview of the modules used for the driving simulator. The simulator was developed for the Chinese market to train learner drivers. The aim of the project was to develop a driving simulator as close to real cars as possible. Further constraints were low target costs and minimization of the conversion work on the vehicle. Unexpected hurdles came from the Chinese administration regarding the acquisition, ownership and recycling of cars. Now follows a description of the hardware and software concepts implemented in the simulator.

6.1.1 Hardware

A major difference to most driving simulator hardware is the use of real cars with minimal modifications. This is possible because of the internal car network technology employed by the automotive industry, the so called Controller Area Network (CAN bus). At the time of the writing, most cars use unencrypted CAN bus communication to allow control units to send and receive various messages like the steering angle, throttle and many more.

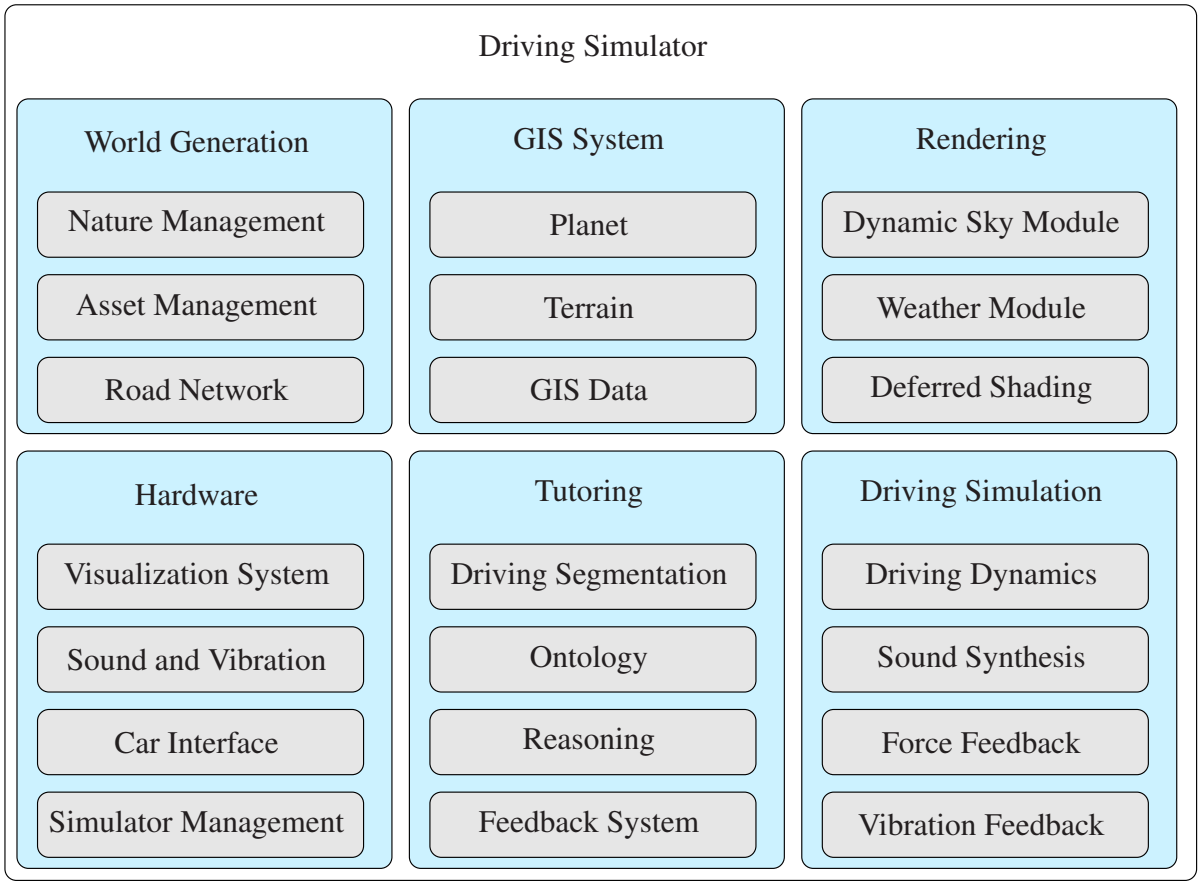


Fig. 6.1: Driving simulator system architecture

Driving Cabin

The first step to access this information and thus turn the car into a user interface is to reverse engineer the communication protocols of each controller. The second step consists in disconnecting controllers that write information such as RPM or speed to the CAN bus. These parameters will have to be computed by the driving simulation and sent to the cockpit controller. This means that one should not only be able to read from the CAN bus, but also to write to it. Each variable that is written to the bus has to be written in the same way as the controller that usually outputs that variable.
This resulting interface is specific for a certain car type and brand. Thus it has to be reproduced if another type of car has to be used. In this project, the interface has been developed for two cars, a smart fortwo and a Jetta. The research and development of the hardware interface has to be attributed to Handfest and Schröder [HSG+13, Sch19].
There are a few aspects that have to be addressed after the reverse engineering of the CAN bus.

1. The steering might feel hard to action as it will not be supported by the engine.
2. Force feedback, for steering or the suspensions, does require additional hardware.
3. Visualisation and tracking systems are also additional hardware.

Steering

Depending on the car, there are different types of steering systems. The ideal case is a system with electrical power steering, because it can be reused as force feedback system. Pneumatic systems would have to be used with a compressor, additional hardware that would augment the costs and the noise level. Another important aspect is disengaging the steering from the car wheels. The best case is if the steering kinematics can be decoupled with a single screw. This also reduces the amount of manual work on the simulator.

Force Feedback

Force feedback can be realized with different types of system. They all have advantages and disadvantages, but most can be discriminated when aiming for a low cost solution.

1. Hexapod
2. Hydraulic suspension system
3. Pneumatic suspension system

A pneumatic solution is the most cost efficient and has also the lowest maintenance costs. None of those systems was included in the final prototype simulator, because of high costs and development complexity.
Instead, a vibration feedback solution was implemented under the driver seat. It allows to give the driver a sense of the driving dynamics. Especially for feeling the difference of the engine when running or not, or to give a basic sense of car speed. The vibration feedback is transmitted using a special subwoofer. The vibration strength is basically the output of a dedicated sound channel. The vibration sound wave is generated alongside the car sound.

Sound System

Realistic sound is an essential component of driving simulators. It drastically improves the feeling of speed and acceleration. Car sounds generated by the simulation are engine noises, air flow noises and tyre on roadway noises. This projects reuses the car speakers to reproduce all those sounds. Installing additional speakers like a 5.1 system has a few drawbacks.

1. A substantial amount of additional manual work is required to install the speakers.
2. The additional speakers may intrude in the cockpit space, thus reducing fidelity.
3. The native sound system of cars are very well configured for the car interior space.

Visualization System

The visualization system for driving simulation is a configurable number of display devices and an optional tracking system. The types of suitable display devices are televisions and ultra short throw projectors. Those displays are arranged around the car to cover as much of the horizontal viewing directions as possible. The directions to prioritize are the front display and then the rear display to allow for a basic driving simulation visualization. The optional additional coverage on the sides extend the viewing field and enhance the driving experience by providing peripheral view and the allowing to look over the shoulder to check for potential vehicles, for example before a lane change. Stereoscopic displays can add to the depth perception and thus to the perception of driving speed. The easiest is to use an active stereoscopic display on a single display device, the front display, thus avoiding the need of frame synchronization. The next possibility is to use passive stereoscopic displays. This also avoids the need for frame synchronization, but does cost much more as the number of projectors has to be doubled. It is possible to use multiple stereoscopic displays without a big impact on the simulator hardware costs, as long as all those displays are connected to a single graphics card. This limits the number of displays to 4 when using currently available high-end consumer graphics cards, as well as a full HD resolution. To actually use four displays on a single graphics card does drastically impact the rendering performance and thus the overall performance. It is possible to use more graphic

cards, but they have to be synchronized to support active stereoscopic displays. The graphic cards that allow such synchronization cost much more and drive the price of the simulator hardware much higher. The last optional and most costly hardware addition to the display system is a tracking system. There are not many options to achieve reliable tracking. Systems of the German company ART do provide such industrial quality tracking devices. The best cost-use ratio is a single component solution like the SMARTTRACK from ART. It is the fastest and easiest solution to deploy and maintain. The tracking device is installed on the passenger seat to cover the full movement space available to the driver. Only the head is tracked to adapt the perspective on all display devices in real time. The SMARTTRACK device has a limited range which one of its disadvantages compared to distributed tracking solutions. This disadvantage does not apply when using such a device in a driving simulator due to the small range of the head movements of the driver.

6.1.2 Software

A driving simulation is a complex interplay of many advanced software modules. As depicted on fig. 6.1, the main software components of the driving simulator are:

1. Open world generation, GIS based
2. Deferred rendering, dynamic sky, day/night cycle, overcast and weather
3. Driving simulation, based on Bullet's vehicle class
4. Ontology and Reasoner modules for the semantic layer and driving tutoring
5. Interfaces and input generation for the simulator hardware components

Hardware Interfaces

The simulator hardware is accessible through a main interface, a custom electronic box with a serial interface. In PolyVR, one can simply use the python serial package to access it. This interface allows to control display elements in the car like the lights, cockpit indicators or speed and RPM gauges, or simply retrieve device states like the pedals, gear lever and steering angle. Setting the state of some elements is important for data simulated by the system, but some cockpit elements simply react to the user interaction like in a real car. For example the left and right vehicle indicators are controlled directly by the user, the related indicators and car lights are toggled directly. The simulation just needs to access the information through the car interface to record the user behavior for his driving performance evaluation.

Driving dynamics

A driving simulator needs a driving dynamics simulation, an interactive simulation of the physical driving dynamics of a car. The relevant components of the car are approximated to the chassis, wheels and the suspensions. This is the low level data structure for the simulation model, as it directly interacts with the collision engine, especially the physicalized terrain, roads and obstacles. Those basic driving dynamics are based on the vehicle class from Bullet [Cou13]. Bullet allows to configure the chassis, mass and collision geometry, the number of wheels, their radius and positions, the physical parameters like the friction coefficient, and the geometric configuration of the suspension system like the position and lengths of the suspensions. The forces acting on the system are applied to each wheel by setting the steering angle, driving force and breaking force. On top of the basic simulation model from Bullet, was developed a full fledged driving dynamics simulation that allow advanced driving behaviour like for example gear shifting, the engine brake or stalling. It is important to accurately replicate the real driving behaviour for more advanced uses of a driving simulator like training learner drivers. This allows to correctly use the car pedals for gearing and experience the overall car behavior like the engine break and stalling, especially when training learner drivers.
This simulation is also the basis for generating the car engine sound and the vibration feedback. The sound is synthesized using an additive sound synthesis approach to create a sound based on a spectrum of typical frequencies. Those frequencies are shifted with the RPM of the engine. The interpolation between two spectra of frequencies is achieved by using a phasor to generate the sound wave. The sound packages are generated and queued as needed. A similar approach is taken for the vibration feedback.

Open World Generation

The world of a driving simulation for learner drivers should be designed to be immersive and realistic, and provide enough space to setup various driving situations. Especially the amount of content required to cover kilometers of street network can get quite huge. On top of that, the systems like the traffic simulation and the tutoring system need the semantic description of the world as their data model. The approach chosen was to generate the virtual world to greatly reduce the amount content creation and to build up the semantic layer at the same time.
Generating a virtual world from scratch is difficult, fully generated worlds have often a lack of natural randomness as algorithms are prone to create some kind of pattern or monotony. One possibility to avoid this consists in not generating everything, but in including real world data. The approach chosen in this work was to use geographic information system (GIS) data as basic structure for the generation algorithms to add more realism to the generated world. Such GIS data can be classified in vector and raster data as described in detail in chapter 2.6. The data used for this project is data available from OpenStreetMap (OSM) [HW08] and from the

Fig. 6.2: Map data from Fuyang District near the city of Hangzhou, China, source: OpenStreetMap

Shuttle Radar Topography Mission [FK00]. OSM provides map data like the street network as a planar graph or the buildings as 2D polygons. The mathematical primitives are thus points and segments, where the segments can form polylines or polygons when closed. The points are used for placing information bound to a 2D coordinate like road signs or trees. The polylines form the road network, they can interconnect to form a graph structure. Polygons are used to delimit areas, especially for buildings, terrain types or water areas. All data can contain meta information like the number of lanes or the building height. This information is attached to nodes, polylines or areas as tags. The system allows custom tags, but most tags are standardized within the OSM community. Tags are proposed and discussed on the OSM wiki [HW08], making the tag system flexible and extendable. The density of information varies widely depending on the region. The reason for this is that the data in OSM is added by users, the more active users in a specific region of the world, the better that region will be mapped. Cities and densely populated regions tend to be mapped with much more details. But this is easily balanced out by the infrastructure for contributing and editing map data. Different tools are available to visualize the OSM map data and edit it, for example by tracing the roads using satellite images. This allows to add or complement missing data according to the project where that data is used, and at the same time benefits the community. The data provided by the DLR is a very detailed height map

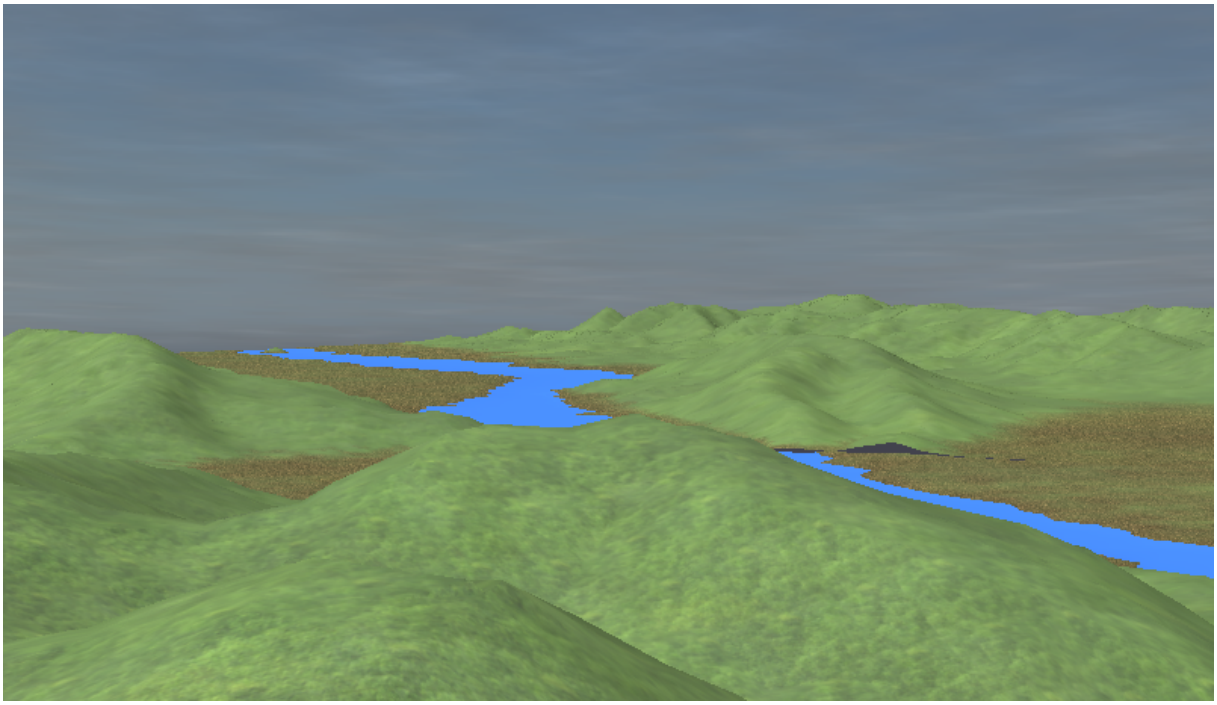


Fig. 6.3: Terrain visualization, topography data from Fuyang District near the city of Hangzhou, China

of the whole world. This kind of raster data can be applied as height maps to model the terrain topography as seen on figure 6.3.

The open world generation starts with the GIS map data. Interfaces for importing the XML based OSM data are implemented in PolyVR as well as a wrapper for the GDAL library, a library for importing raster formats used for topography data. The top most data structure is the planet module, it handles polar coordinates and converts them back and forth to the Cartesian coordinates used for the virtual environment. It also manages the planet surface, split into small chunks to allow to dynamically load and unload them as needed. The planet module is further described in section 4.2.1. The next important module is the terrain module. It allows to visualize large scale topography in real time. It uses tessellation shaders to apply the topography data or height map as vertical displacement map. It also enables interacting with topography data by ray casting through it. This is important for basic navigation or interaction. Another important role is to provide an interpolated height value and surface normal for placing objects on the terrain.

The terrain and the nature modules are the basis of a virtual open world, but for a driving simulator one needs urban elements like roads and buildings. Buildings are quite easily generated as the geometry can be kept very simplistic while achieving a high degree of realism. The quality of the building visualization mainly depends on the facade textures. It is important to have a system that allows for a high variety of facades but also heavily optimize the rendering as a virtual world can contain hundreds of buildings. The streets on the other hand are highly complex to generate, especially regarding intersections and crossings. Based on the graph nodes there

are virtually many topological possibilities in which roads can come together in an intersection. The angles are not discretized, the algorithms that generate the intersection geometry have to be very robust and well tested.

The world generation creates a complex road network with lanes, intersections and parking lots, but it lacks something very important, traffic. Traffic is a very complex subsystem of the open world generation. It is a complex simulation of particles and flows, as each car, bicycle and pedestrian is traversing the road graph based on their goal, traffic rules and personality traits. The complexity of the simulation is also increased due to the interaction with the user. The traffic simulation is an interactive simulation, thus requires many optimizations to run with near real-time performance. The simulation is split into two layers, a macro simulation that uses the road network graph as simulation model and only simulates flows on graph edges, and a micro simulation that simulates particles moving on the road graph. The micro simulation is only active in a small area, usually centered around the user. This area can dynamically change in size and its position to follow the user. The macro simulation is a flow simulation. Each edge of the road network has a density value that changes over time. Edges with high density will transfer to edges with lower density. The micro simulation is essentially a logistics simulation. Each entity has a current position and a goal and the restrictions imposed by the roads network and traffic rules to reach it. In addition to the road network, especially road lanes, intersections, traffic signs and lights, the simulation model defines vehicles. Vehicles can drive along a road lane, turn at intersections and switch lanes to advance on the graph. They respect the traffic rules like stopping at red traffic lights, waiting for other vehicles that come from the right, and overall respect speed limits and avoid bumping in other vehicles. The simulated vehicles respect the user and avoid bumping into his car if possible. The implementation of the traffic simulation has been done in collaboration with Sebastian Friebe and Felix Michels. The first early implementation has been done in 2015 [Fri15]. After major changes to the world generation, especially a high amount of enrichment of the GIS data, the simulation had to be redone from anew. the second implementation was quite rudimentary, and thus was greatly enhanced by Michels in 2019 [Mic19]. Michels further refined the road network system, added traffic signs and lights. He then worked on the vehicle behaviour and parallelization of the simulation.

An important part of the scene is usually occupied by the flora. Realistic looking trees are a key element of an immersive virtual world. Trees for instance are created using an algorithm to generate the armature and a set graphic shaders to draw the hull. The advantage is to be able to easily deform the tree, like through wind effects or cutting branches, with little impact to performance. This also facilitates optimisations like automated LOD generation.

The first LOD is a reduction of detail in the hull computed with the shader. The second LOD is reducing the amount of detail of the armature. The last LOD is not based on single trees, but

actually computes simple mock-ups of each tree and agglomerates it into a single geometry. This requires an extra module in PolyVR named VRWoods.
This module allows to add an arbitrary number of trees which are kept internally in an octree structure. The trees are in the leafs of the octree, and each level of the octree contains another LOD level, each octree node contains the aggregated LOD geometry of all the trees below it. The same system also handles bushes of all kinds. The grass, including flowers and herbs, is managed by a separate but very similar system.
The open world generation allows to instantly create large scale virtual environment based on any available map and topography data. This is an important asset for many applications, from driving simulations to validation scenes for autonomous driving.

Rendering

A driving simulation needs to be able to efficiently handle many complex light sources in the virtual environment. The following light sources are implemented:

1. The sun illuminates the scene during day cycles
2. The car headlights are controlled by the user
3. Many streetlights can be active in the scene all at once
4. The cars from the traffic simulation have all headlights

It is important to be able to illuminate the scene with all those light sources, as well as visualize the shadows thrown by the most dominant light sources. The only way to achieve this is to use deferred lightning, a deferred rendering technique that splits rendering in multiple steps and postpones the lightning calculation.

Tutoring System

The tutoring system is responsible for taking the user by the hand. It records, segments, classifies and analyses every interaction of the user and aggregates it into his profile. The system can then generate tasks for the user based on his profile, thus creating a personalized and optimized learning curve. To the user it takes the role of a virtual driving trainer. gives him hints, explanations and even context sensitive theoretical background information. The tutoring system is mostly based on the ontology system in PolyVR. Many ontologies are the basis for the system, for instance ontologies with didactic concepts, concepts of street networks and traffic and even concepts about driving dynamics to mention just a few. The user, his profile, the car state and his environment are entities of the semantic layer. With each frame, the data is actualised from the user interaction and simulations like driving dynamics and traffic. The data is then used by the reasoner to deduce driving errors, task progression and user skill increase.

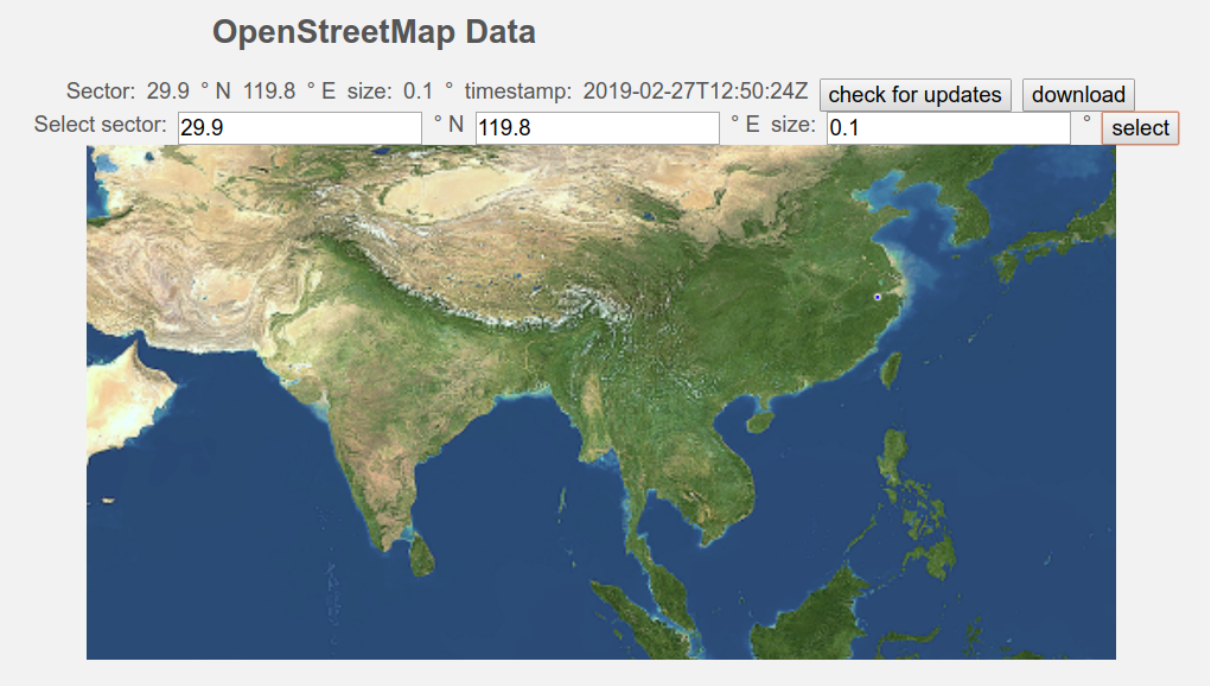


Fig. 6.4: Management of map data chunks acquired from OpenStreetMap

Content Management

The content management is especially important for an open world environment. Map data editors enable easy editing of the world assets like the road network, the buildings or biomes. The map data has to be split in chunks and delivered as needed to the simulator. A web server has been setup to manage the map and topography data as seen on fig. 6.4. This allows to control the map updates, preventing the use of corrupted data, and avoid unnecessarily straining the OpenStreetMap server.

6.2 Plant Engineering

The second major virtualization use-case of this thesis is the virtualization of production lines during their product development process for the purpose of development validation, training and more. Integrated production lines are complex machines that handle the material flows, process or machine materials, do quality checking and package the final product, all in one automated system. They are built for many producing industries so that each comes with their specific requirements. Small and medium sized enterprises usually specialize in a specific domain and type of plant. The domains relevant in this chapter are the food processing industry, especially chewing gum production, industry scale balancing machines, especially for tyre on rim mounting, as well as Siemens automation systems. Complementary types of machines are used to package the products like blister machines or packaging machines. The final production lines are often assembled with machines from different engineering enterprises, depending on the product, production process and specialization of the companies.
Those machines are themselves products, developed in a typical product development process. As they are highly complex and costly products, usually lot size one, their product development process is the perfect application for virtual mock-ups.
From the perspective of virtual mock-ups, such a plant consists of mechanical components, rigid and flexible parts interconnected in kinematic chains. The production needs an intralogistics and process simulation engine as well as various numerical processing simulations. Such a complex virtual reproduction of a production plant is called a virtual twin.

6.2.1 Virtual Engineering Application

To validate the methodology proposed above, an application has been developed using the VR framework PolyVR. This design review application implements the workflow as described in section 5.5.1, and adds a set of tools, especially useful in the VR environment with its 6 DoF interaction capabilities. The basic interaction with the CAD model is to explore it using various paradigms. The first is to drag and drop the geometric parts, where the snapping engine allows to easily put them back into their original place. The second tool is the clipping plane, it allows to explore the model layer upon layer. The final couple of tools, to help explore the model, are to change the appearance of the geometry materials, either by turning them semi transparent or fully invisible. Once specific issues have been identified in the model, it is necessary to communicate them to others. This can be through some kind of documentation, or by directly resolving the issues and applying necessary changes to the model in the CAD system. The VR application supports those activities in a number of ways. First the possibility to take screenshots allows to easily document the issues, and then a function to set way points places markers on the ground which allow to store the camera pose and quickly recover it by simply clicking on such a marker. The selection tool allows to quickly communicate the relevant part to collaborating users,

especially through the bidirectional synchronization with the selection in the CAD system. Additional information is presented on the selected object, its name and its position in the product structure. The viewer for the product structure also allows to select nodes of higher hierarchy like a sub assembly, and to interact with it by toggling its visibility.

6.2.2 Automation Systems, Simatic S7

Large facilities such as power plants or production plants are highly automated. The important systems are monitored and controlled from a control room. This greatly reduces the number of maintenance personnel, but also requires complex automation systems to continuously run the whole plant and avoid downtime as much as possible. Those automation systems are critical components that are connected to all actuators and sensors. Lintala et al. [LO13] show for example the importance of security aspects of such systems and propose enhancing them by integrating functional safety information into design requirements.
When virtualizing a plant it is imperative to emulate the automation system to reproduce the authentic behaviour in virtual reality. This is especially important for interactive use-cases like training maintenance personnel. In the following we describe the virtualisation of a Siemens Simatic S7 system for a training application. This was implemented as part of an industrial project with Siemens Power Academy in Karlsruhe. To validate the application, a simple training scenario was defined, the maintenance of a redundant system with two sets of a PS 407 10A power supply module, a CPU 410-5H module and a network module CP443-1 (fig. 6.7). Even such a small system requires a few steps to diagnose and repair a malfunction, either by replacing a part or component or by fixing a wrong installation. Various errors include wrongly connected cables, a defect CPU module, a missing extension card in the CPU module, or a wrong configuration of the CPU rack lever at the back of the CPU as depicted on fig. 6.8.

Virtualisation

The virtualisation workflow is depicted on fig. 6.5. The data needed for the virtualisation are 3D models of the Simatic S7 modules, assembled as seen on fig. 6.6, as well as a description of the behavior of the modules when interacting with them. The geometries of the S7 series are mostly openly available from the Siemens website. The available data for each model varies, some include only a picture or 2D layout as DXF file, and some include the CAD hull model as STEP file. The CAD models have to be tessellated and refactored. We use FreeCAD and Blender to prepare the VR models. Textures were added to enhance realism, for example does using baked in ambient occlusion maps greatly enhance the realism of the virtual models. Missing geometric features have to be added, especially those relevant for the training scenario. Interactive elements of the geometries have to be manually segmented and classified like buttons and levers, but also indicators like individual LEDs. The result is shown on fig. 6.7.

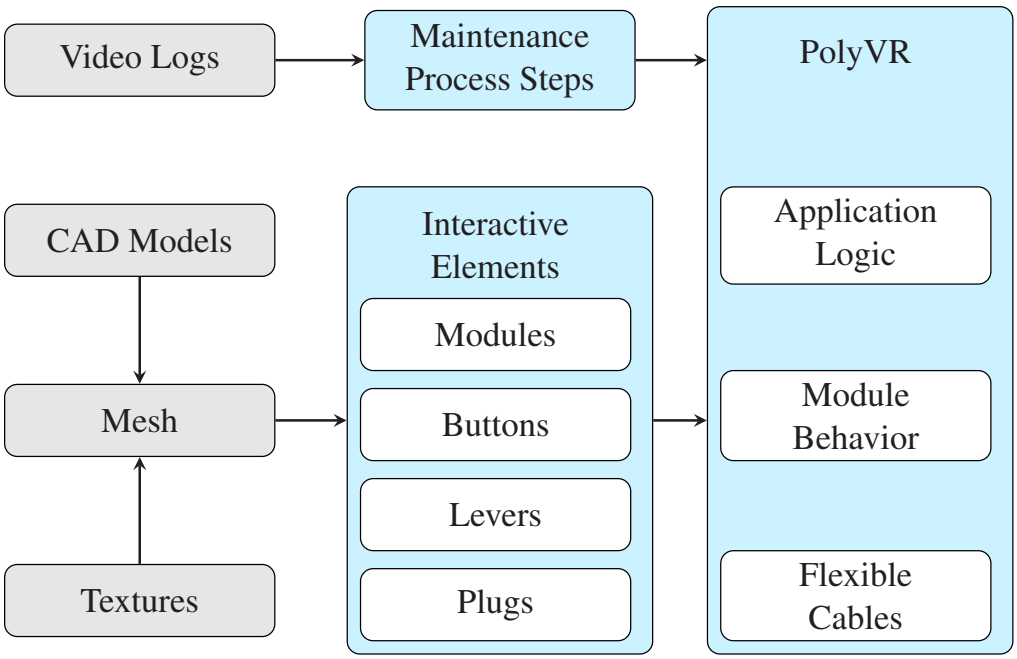


Fig. 6.5: Virtualization workflow of the S7 VR model

After the preparation of the 3D models, further virtualisation is done using PolyVR. The cables are flexible geometries that connect two plugs. They are realized using the VRStroke module in PolyVR. This allows to generate the cables and remake them after each change. The cables are fully interactive by dragging one of their plugs. Another system employed is the VRSnappingEngine. This PolyVR module allows to define the key points for snapping the geometries in the virtual model. This is essential for the drag and drop interaction paradigm, it allows the user to interactively unplug and plug cables, unmount and mount the S7 modules on their rack, or configure the various small parts like the batteries in the power supply module. The snapping provides a clear feedback if two components are connected or a component is at its correct position. It also signifies to the user that the application is aware of his intention to connect two components, as the application is informed by the snapping module via callback functionality.

Application Logic

The key requirement for the training environment is the realistic behaviour of the modules, especially the CPU. The best way to integrate such behaviour is to connect an emulation of the S7 CPU operating system as a back end to the virtual model. But due to the limited scope of the project, another approach has been chosen. The visible behaviour is limited to the LED indicators on the front of the S7 modules. Thus it is possible in this case to script the sequences of the LED patterns according to the configuration of the modules. This is possible because the amount of combinations of connected cables lever positions is not too high. There are two possible states for the power supply, on or off. The configuration on the back of the CPU has

Fig. 6.6: Example of a real PLC, modules are slightly different from the models that were virtualized, source: [Sie]

four additional states, two for the rack ID lever and two for the expansion card, present or not. The two cable slots on the front left of the CPU have three states each, not connected, wrongly connected, and correctly connected. Amongst those 9 cases, two can be discarded because if one of the orange cables is correctly connected, the other cannot be connected in a wrong manner. That leaves us with 7 states. Those are the elements relevant for the CPU maintenance training. Their states add up to 2x2x2x7 = 56 combinations. This amount of combinations is low enough to allow the explicit scripting of the module behaviour as it has been done for this use-case. But this also shows that with each additional configuration option the amount of combinations grows so fast that scripting the behaviour is becoming nearly impossible.

Training Application

The training with the virtual environment is split in two phases. An exploration phase, where the user can navigate through all the steps of removing the CPU module and the installation of a new CPU module. Each step is animated, and hovering over each component gives the user an excerpt from the official manual. The steps are depicted on fig. 6.9
The second phase is to train the maintenance by solving the problems based on different initial states of the system. The user has to interpret the LED indicators and find the problem. The goal is to fix the problem and to reboot the system in its normal running state as seen on fig. 6.7.

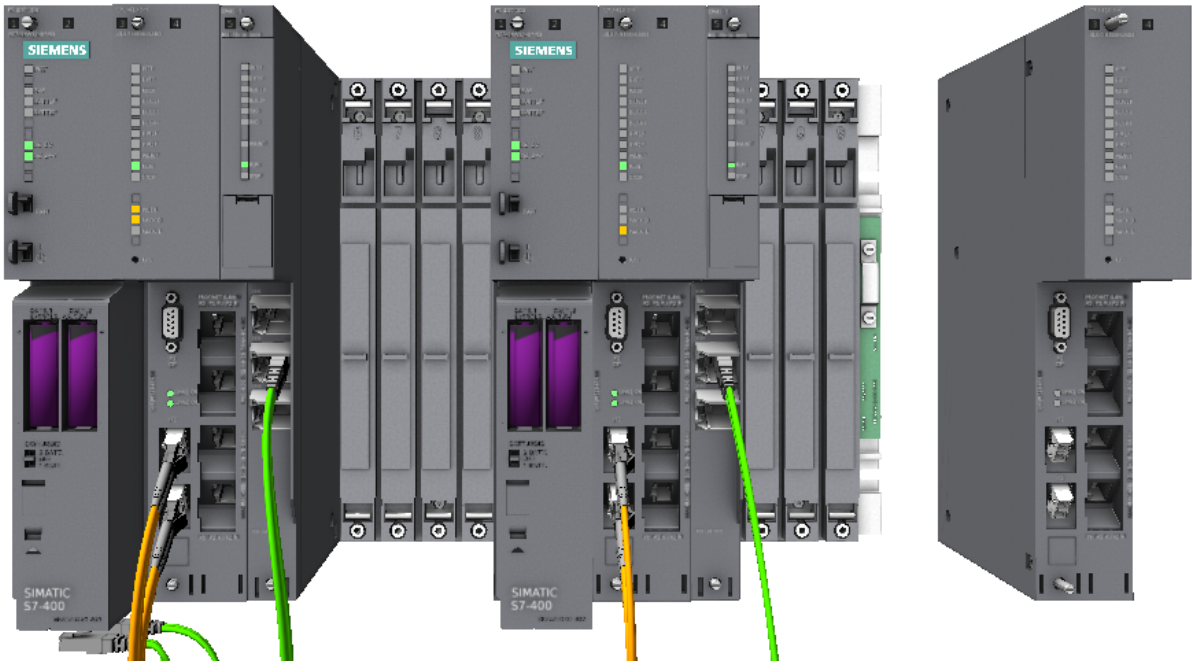


Fig. 6.7: Virtual model of the S7 automation server, PS 407 10A, CPU 410-5H and CP443-1.

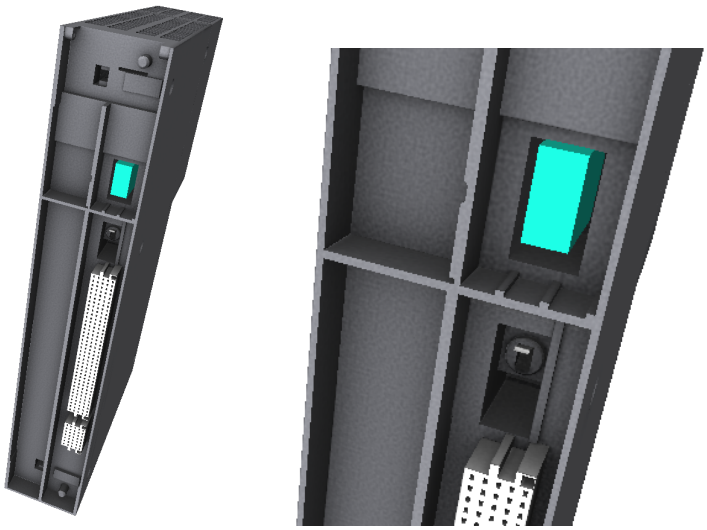


Fig. 6.8: The back side of the CPU, important are the green expansion card and the position of the lever below the card.

Conclusion and Outlook

The project described above is a typical example of a classical virtualization. The use case is very concise which allows a complete implementation with every possible system state. The important aspects for the thesis are the usage of PolyVR. The whole project was realized in a month of time, where at least half of the time was spent on gathering the data and refining the 3D models. The scripting was minimal as standard modules like the VRStroke or the VRSnappingEngine allow to quickly setup the interactive elements of the VR model. The python

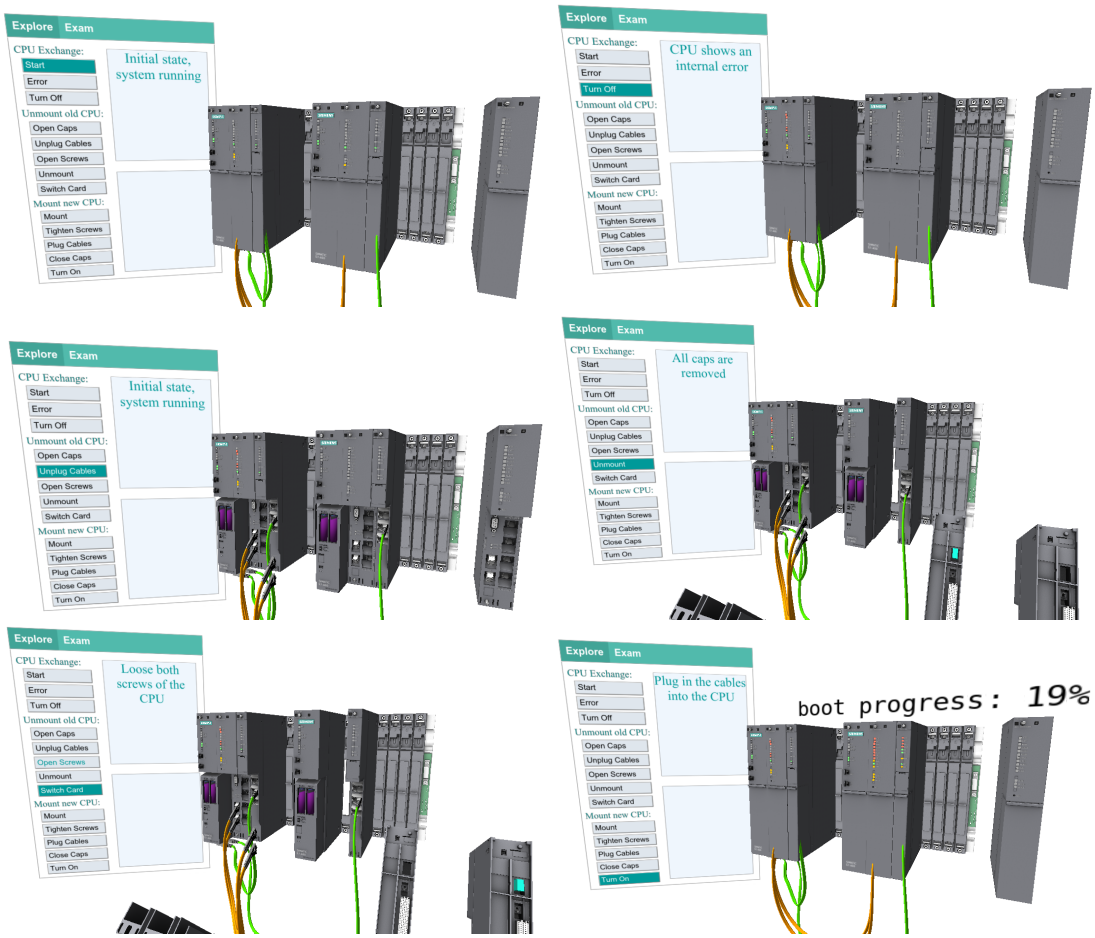


Fig. 6.9: Some of the major steps to exchange the CPU module.

scripting also greatly simplifies the implementation of the component behavior, especially the sequences of the LED indicators. This examples shows the extensive capabilities of PolyVR to quickly virtualize a training environment for industrial use.

Further developments will include an S7 emulation back-end. This will avoid the manual scripting of module behavior and thus avoid missing behaviors for complex situations.

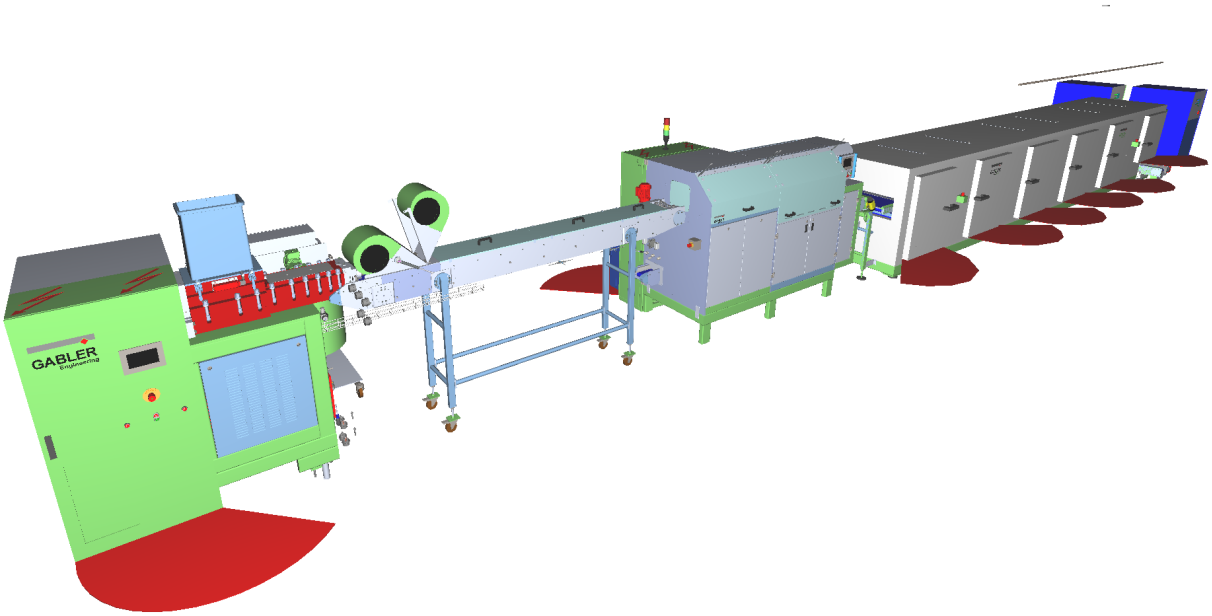


Fig. 6.10: Integrated production line for bubble gum from Fa. Gabler

6.2.3 Automating the Virtualization Workflow

The virtual engineering method as described in chapter 2.7 is an extension of the product development process. It aims at greatly reducing the duration of each iteration of the construction process by introducing design reviews using virtual reality technologies. To implement this method it is important to setup a fully automated virtualization of the product development data. The methodology and implementation of the virtualization process has been presented in chapters 4.3 and 5.5. This section will show how to setup such an automated virtualization for integrated production lines. The data used in this section is provided by the company Gabler Engineering GmbH, located in Malsch, in the state Baden-Württemberg, Germany. The use case is a integrated production line depicted on fig. 6.10. This part of the work was developed in collaboration with A. Benedix, Q. Vllasa and A. Hristov [Ben19, Vll19, Hri19].
The main challenge of automating the virtualization process lies in the different domains that have to come together. First, there is the mechanical construction data, then there is the electrical data, the model of the wiring of the electrical components, and last there is the programming of computation units. This chapter shows how far the virtualization process of the production line can be automated. Especially the import and integration of all the heterogeneous data sources will be addressed as well as the construction of the virtual model of the production line in just a few minutes.

Mechanical CAD

The first step of the virtualization process is to import the mechanical construction data, the geometric representation of the model parts and the product structure that defines how the parts

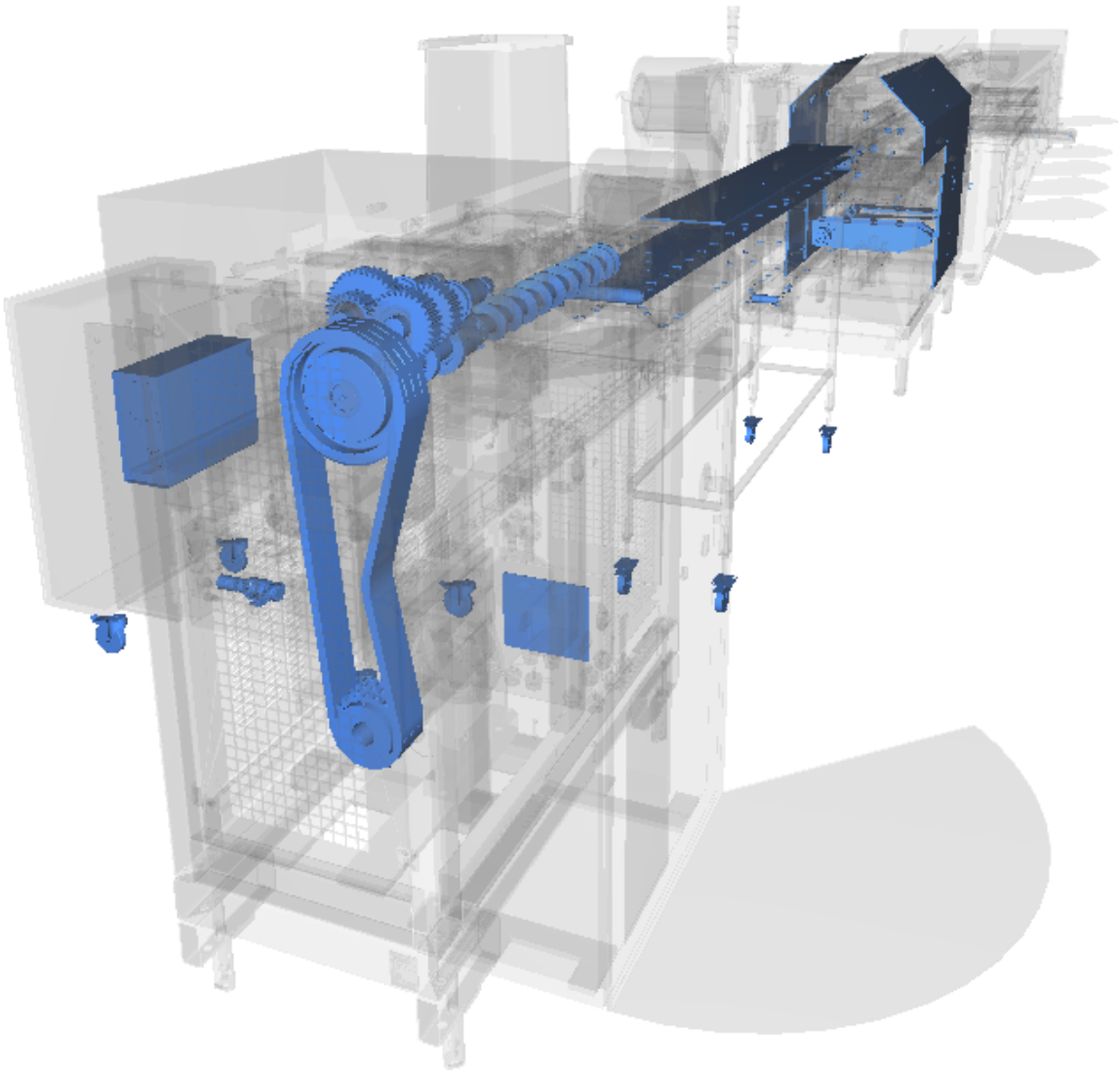


Fig. 6.11: Kinematic parts highlighted in blue

are structured hierarchically and positioned in space. The MCAD data can either be transmitted using the STEP format or by using a plugin in the CAD tool. For this use case a production line was available in STEP format. Further information that may be modelled are dynamic information like constraints and kinematics, but those are rarely specified and furthermore cannot be transmitted by any common exchange format, requiring a plugin to extract that information directly from the CAD software. When available, dynamic information can be directly utilized, but in most cases the dynamic information must be added to the static model. This is a very complex task because of the way the CAD model is constructed. The CAD construction is aimed at being read and interpreted by the shop floor engineers who machine the parts and assemble the production line. There are many information that are not explicitly modelled and solely rely on the aptitude of the engineer to interpret the 3D model based on his experience and

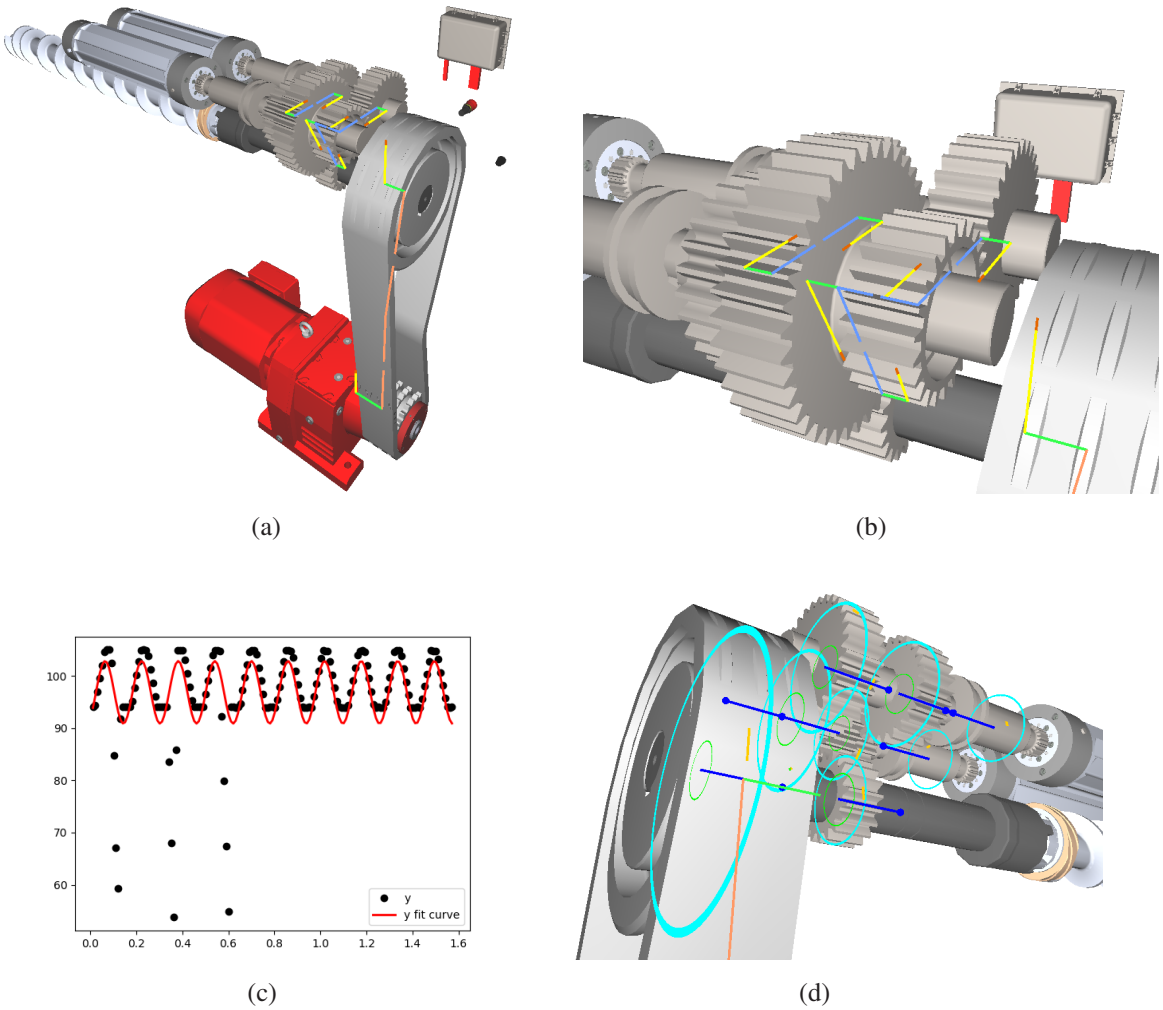


Fig. 6.12: Mechanism, a) kinematic chain from engine to screw conveyor, b) chain reconstructed based on geometric analysis, c) analysing the vertex polar coordinates by fitting a sine to obtain gear angular pitch, d) gear parameters extracted from geometry analysis and used for mechanism simulation

knowledge. This is the only guaranteed foundation for the virtual model, the static geometric data that is consistent only if interpreted by a human person.
The challenge at this point of the virtualization is to automate this crucial step. In order to do this, the semantic layer, ontologies and reasoning, as well as geometric analysis can be used to reconstruct the kinematic chains of the machine. The necessary steps are as follows:

- Classify every component
- Analyze component geometry
- Compute simulation parameters
- Assemble the topology of the components
- Simulate the dynamic behaviour

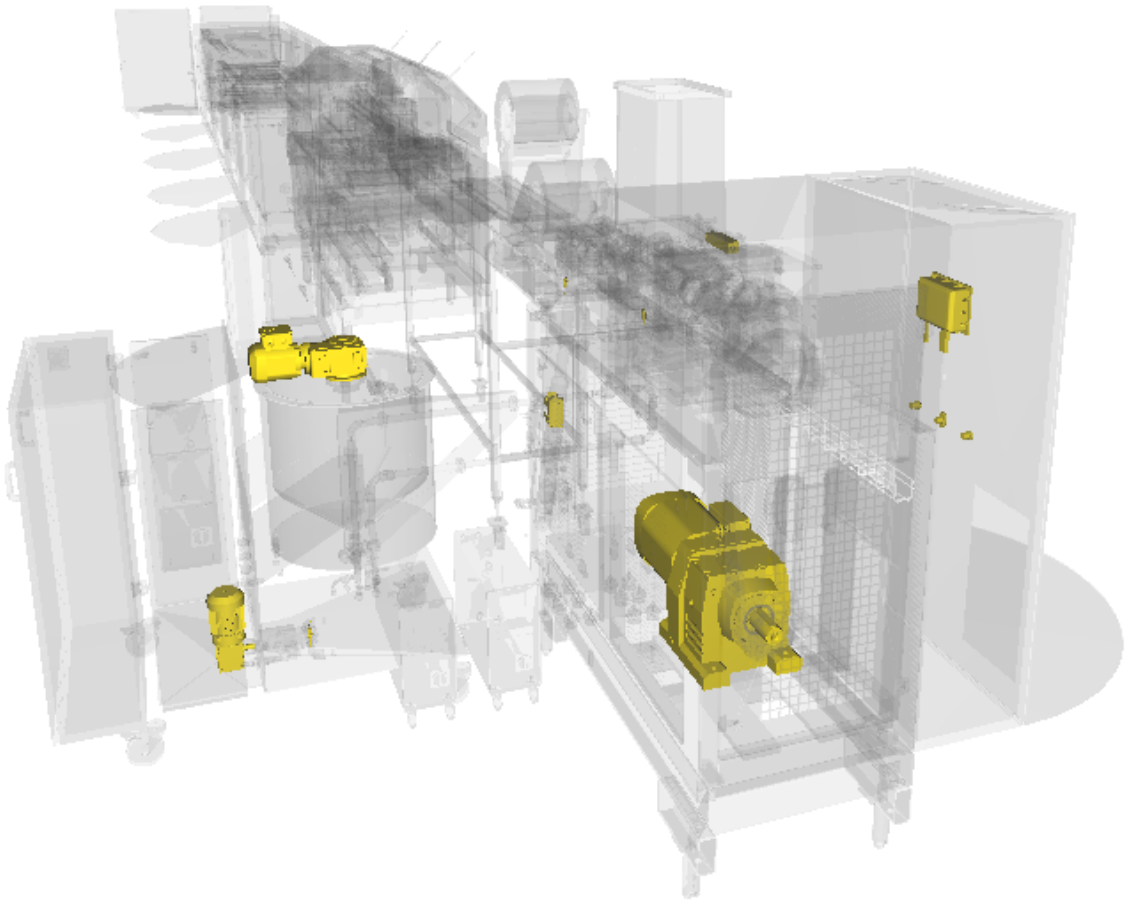


Fig. 6.13: Electrical parts highlighted in yellow

The classification of components is quite rudimentary as we only use the meta data of the CAD parts, their names. In some cases the object name contains the type of the object like gear, screw, plating etc., if this is not the case then at least an article number allows to identify the object type. Figure 6.11 shows the production line and some dynamic components highlighted in blue. Notable are the gears of the mechanism in the front, the conveyor belt in the middle section and the hatches in the back. When classifying each part, an entity is created in the semantic layer based on a corresponding concept in a generic ontology for mechanical parts. Once classified, the geometry can be analyzed to extract basic properties like the orientation of a gear shaft, or the degrees of freedom of a joint. Figure 6.12 shows the analysis results of the mechanism of the extruder module. The electric motor drives a belt, the gearing and the screw conveyor. The topology of the components is essentially described by the connection between components, joints or fixations. It is important to analyse the geometries to extract the necessary parameters. For example a gear has a rotation axis, a radius, angular pitch and teeth size. All those parameters can be computed as described in chapters 4.3 and 5.5.5. Figure 6.12 shows the computed parameters for each gear as well as the kinematic chain. Ontological data and

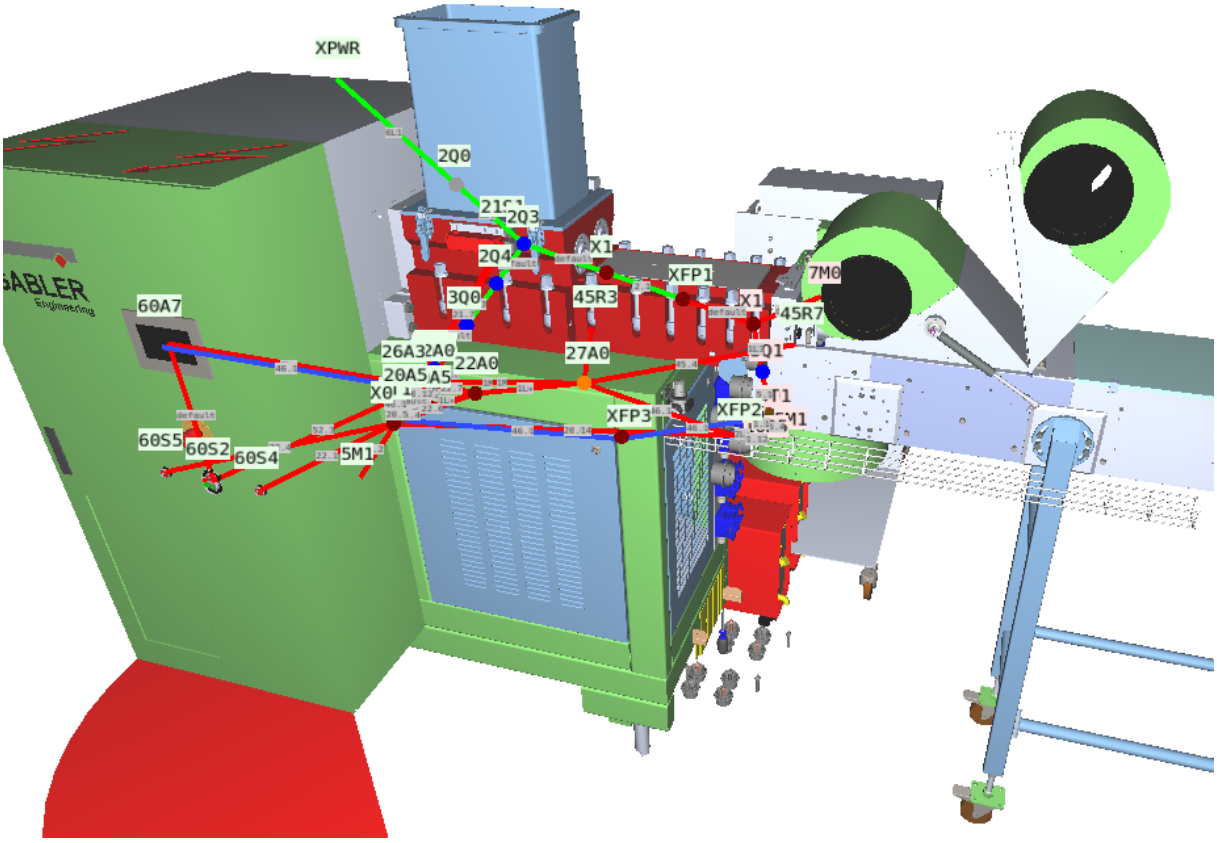


Fig. 6.14: Extruder with touch panel, switch and other electrical components

rules can help to recreate the topology. For example, it is likely that a gear is attached to an axis if they are concentric to one another. Once the neighborhood of the mechanical component is reconstructed, it is possible to deduce the basic dynamic behaviour, how can it move, how does it affect the movement of its neighbors and how the later do affect its dynamics and so on. The final step is to assemble the kinematic chains. A dedicated simulation module in PolyVR handles the interactive kinematic system and the consistency of its dynamic behaviour. The belt and gears are simulated with the PolyVR mechanism simulation and the gear on axis connection is a fixed joint simulated with the PolyVR kinematics simulation. Further information may be available when incorporating the ECAD data into the VR model.

Electrical CAD

The second step of the virtualization process is to import the electrical CAD data and merge it into the geometrical mechanical model. The CAD system used by Gabler is the EPLAN software. It allows to easily export the plans with the electrical components and the wiring. The result is a planar graph with nodes corresponding to various electrical components like clamps, switches, terminals, PLCs and much more, and with edges that correspond to electric wires or data wires. Each electrical component has a unique ID, also used to model the connections.

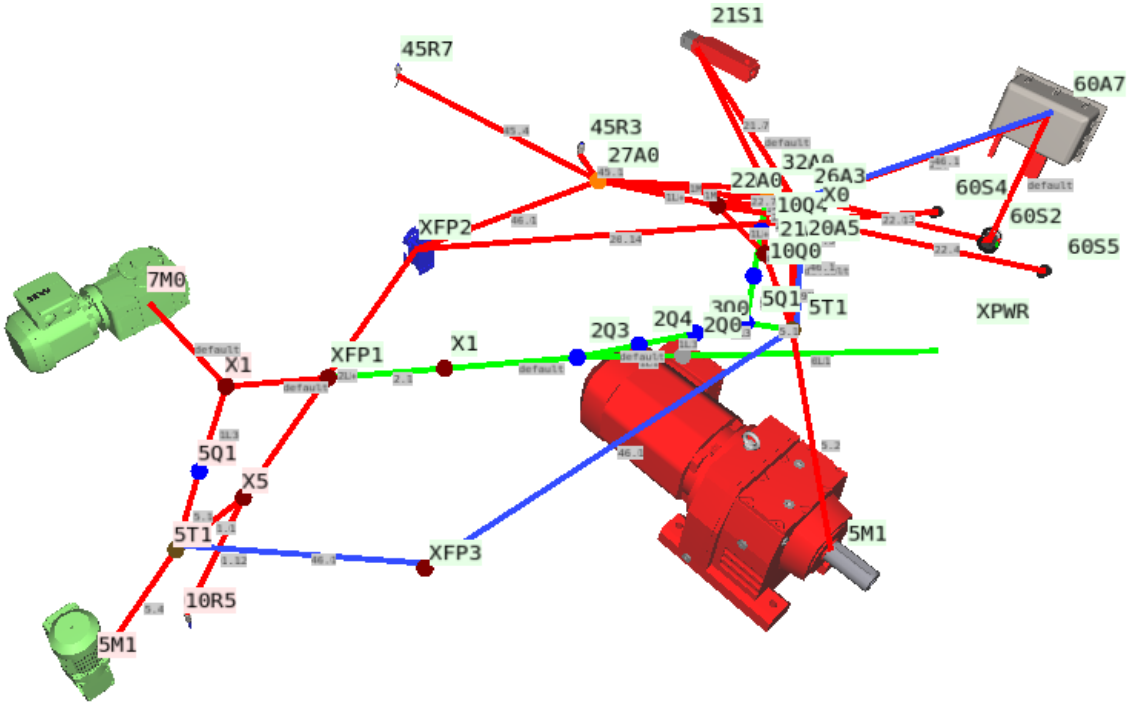
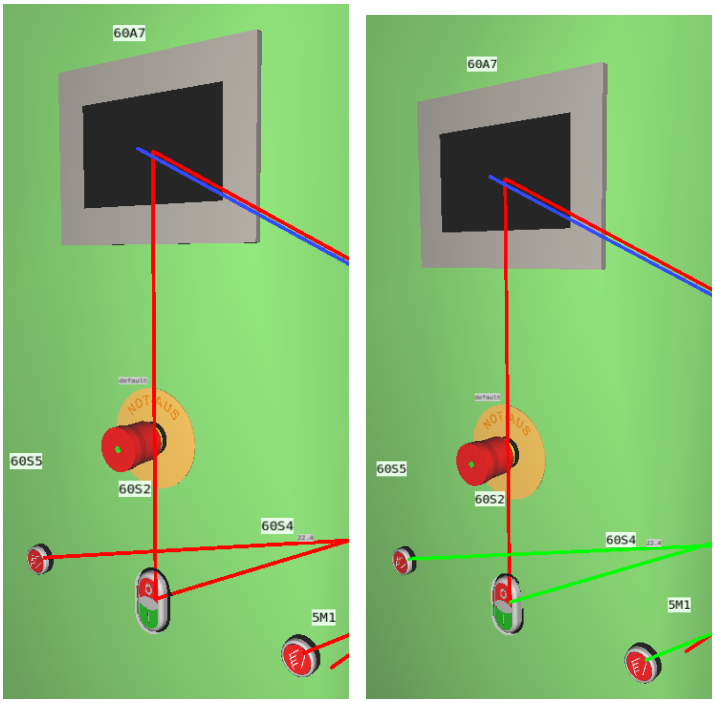


Fig. 6.15: Visualization of the wiring, green and red the electric wires and blue the Profinet bus

The main difficulty is to map the unique IDs in the ECAD data to components of the MCAD model. This is done again using the meta data modelled in the ECAD system. Usually, there are string information that are found in the MCAD and ECAD data that match. Figure 6.13 shows the Gabler production line with the electric components, that have a geometric representation, highlighted in yellow. They are quite sparse and mostly consist of actuators and HMI elements like the touch panel and switches. Once mapped, the wiring graph can be visualized in VR as shown on fig. 6.14. This is done as a graph visualization, the layout is automatically generated using the 3D positions of the components in the subset overlapping MCAD and ECAD components. A simple spring based algorithm enhances the graph visualization. The figure also shows the HMI elements and how they are connected with the wiring, in red the electric wires and in blue the Profinet bus. Figure 6.15 shows only the electric components and the wiring. Those are only a subset of all the electric components, the ones that have a geometric model and those that lie on paths between them. The green wires are electric wires with an electric current.
The final step of the ECAD integration is to simulate the behaviour of the wiring, the signals, currents and the electrical components. The simulation changes the state of the wires and components according to user interaction with the HMI components. Pressing on the power switch lets the current flow from one side to the other as seen on fig. 6.16. The change of current is a signal that gets transmitted over the wiring to the IO modules of the PLC. The simulation method and implementation is described in chapters 4.3 and 5.5.6. The last missing piece to



(a) (b)

Fig. 6.16: Extruder power switch, a) switch not pressed, no current on the red wires, b) switch pressed, current flows over the green wires to the PLC

this signal chain is how the PLC handles the signals. This is modelled in the programming of the PLC, described in the next section.

Automation Programming

The third step of the virtualization process is to emulate the programming of the automation modules. The development of the PLC programming is usually done by yet another department next to MCAD and ECAD development. Those engineers will only work on programming and parameterizing the PLCs. The programming defines the behaviour of the PLC, how it processes input signals and control actuators. The state of variables can be changed by corresponding analog or digital inputs from the wiring system. Those inputs may be set from sensors or HMI components connected to the PLC through the wiring system. The PLC uses the input variables to compute the output variables, often taking additional HMI variables into account. The programming sets those variables as defined by the programming flow. The output variables define currents at the PLC output modules. The wiring connects those outputs to other electrical components, changing the behaviour of actuators. Figure 6.17 shows part of the programming of the Gabler line responsible for the extruder module. Pressing the power switch changes the state of the variable Button_Ext_start. This induces the signal to travel through the programming logic and start the extruder motor. This can be seen as long green line in the central column, third row.

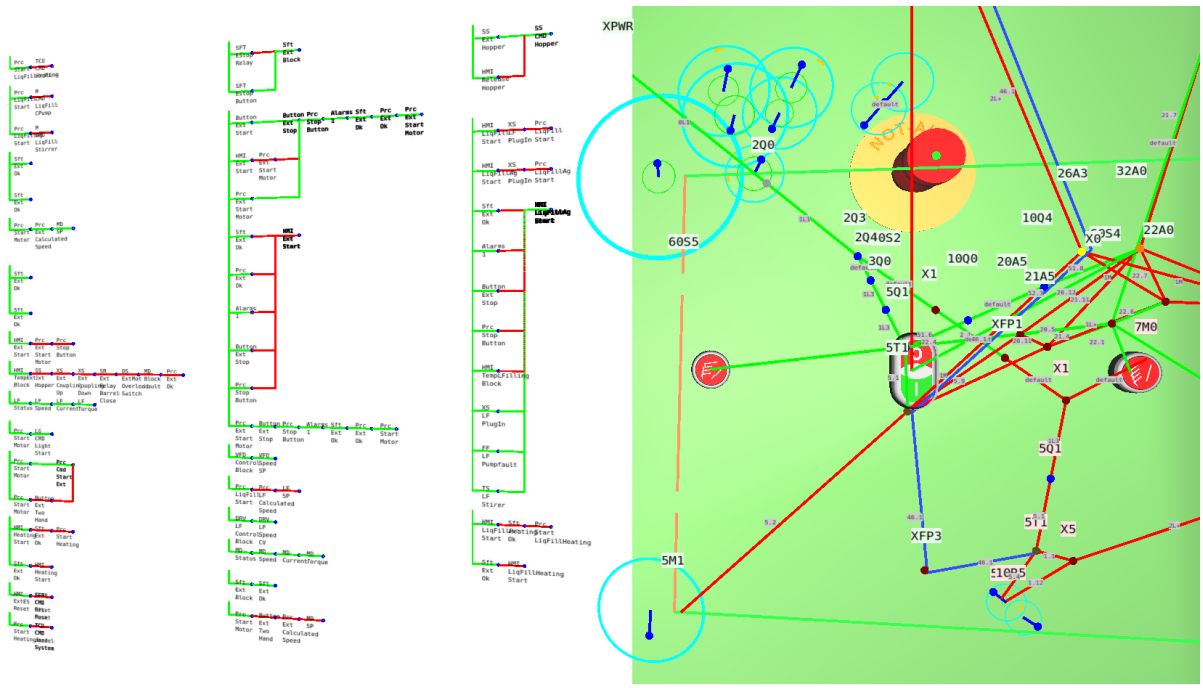


Fig. 6.17: LAD program emulation

The emulation of the programming is achieved according to the method described in chapters 4.3 and 5.5.6.

The modules of the production line have configuration touch panels to configure the behaviour of the PLCs and thus of the machine. The software running on the panel features a graphical UI with common widgets to change parameters, for example to change an actuator speed. It is important to virtualize the touch panel as an interactive 2D interface in VR. This is done using the website rendering module from PolyVR. The software and its UI on the panel are replicated as a website. The machine specific programming and configuration logic can be loaded in a generic manner, making the virtualized panel a generic emulation environment. The panel exposes the parameters that the user can change. This completes the chain of functionality of the virtual model to allow the user to start and configure the machinery, paving the way to many applications.

Conclusion and Outlook of Virtualization Process

A further virtualization step is to take BIM data into account if available. This is necessary if the machine to virtualize has a significant interaction with the surrounding building it is deployed in. This can simply be a matter of space or logistics, but can go as far as interfacing with the building infrastructure, hydraulic or gas pipelines, high voltage electricity or waste management. A common BIM exchange format is IFC. Current building CAD software like Revit can export IFC. IFC is a BREP file format and contains the meta data associated to the building elements. PolyVR supports the import of IFC and uses the OCE library to tessellate

the BREP models. This allows to easily integrate the production line in its surroundings and plan accordingly.

To fully automate the CAD to VR virtualization workflow, taking into account MCAD, ECAD and automation, would lead to a fully functional virtual twin of the product in just a few minutes. To achieve this would be the holy grail of the virtual engineering method as validation iterations of CAD development would get drastically smaller and more efficient. This work is a step towards this vision, slowly inching towards a fully automated virtualization process. Figure 6.18 gives an overview of how far the automation of the virtualization process has been implemented. The major issue that is still open is how to handle the lack of semantic information to fuse the CAD data from the different domains together. Every system does provide the means to identify the component, but there are no identifier used across all the systems. The method used in this work relies on the fact that there are no ambiguities, this means that the data is modelled in a way that it is consistent for the interpretation by a human. The system uses geometric analyses, ontologies and simulations to integrate the heterogeneous data and build the virtual functional model, but if this fails, the point will be reached where the missing information has to be added in the CAD system. The goal is to push this point as far as possible to avoid additional hurdles for SMEs to deploy such technology. Another major outlook is to further extend the use of ontologies and reasoning as well as extend towards other domains like for example hydraulics and pneumatics. On one hand, it is necessary to achieve a more robust classification of components in the CAD data, on the other hand extend the simulation of mechanical and electrical components. It would be possible to use the reasoning system to even automatically parameterize processing simulations.

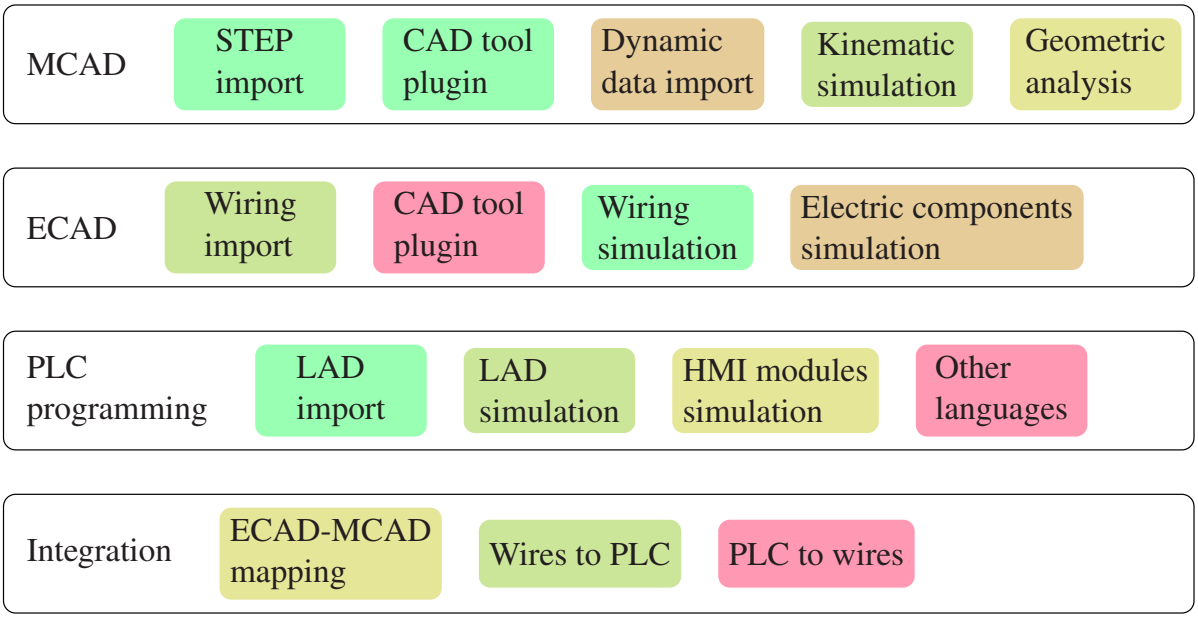


Fig. 6.18: Estimated progress of automating the CAD virtualization process based on Gabler line

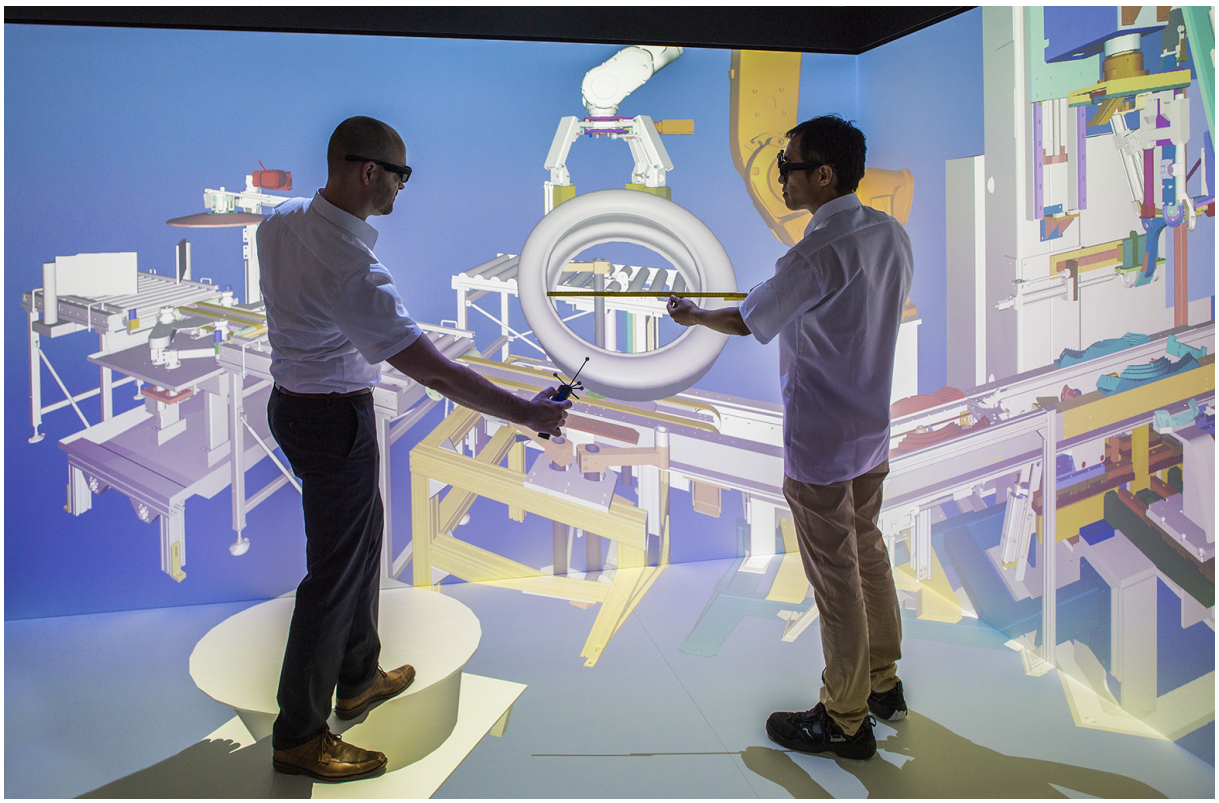


Fig. 6.19: Designreview at Schenck RoTec GMBH, 3-sided CAVE, source: [RoT16]

6.2.4 Deploying the Virtual Engineering Method in Industry

An important goal of this work is to enable SMEs to use virtual reality technologies to help in their product development. Up to this point, the research and development of the necessary virtual engineering software toolset has been described in detail. The resulting VR authoring software PolyVR can configuring complex distributed visualisation systems and implement CAD virtualisation workflows. This allows to implement the design review based on the virtual engineering concepts in the SME processes.

An excellent opportunity to validate this came with a request of a company to commission their new CAVE installation and deploy a design review system. The design review application has been deployed at the Schenck RoTec GMBH, part of the Dürr AG. This company develops fully automated integrated production facilities for the balancing of rotary parts on industrial scales. Their VR hardware setup consists of a three-sided projection system as depicted on fig. 6.19. It is a low cost CAVE system with top projections using state of the art UST F50 Barco projectors. The company has about 50 MCAD construction engineers and aims at establishing the CAVE system as design review collaboration space for their product development process, especially for regularly validating the progress of the product development and communicating with customers.

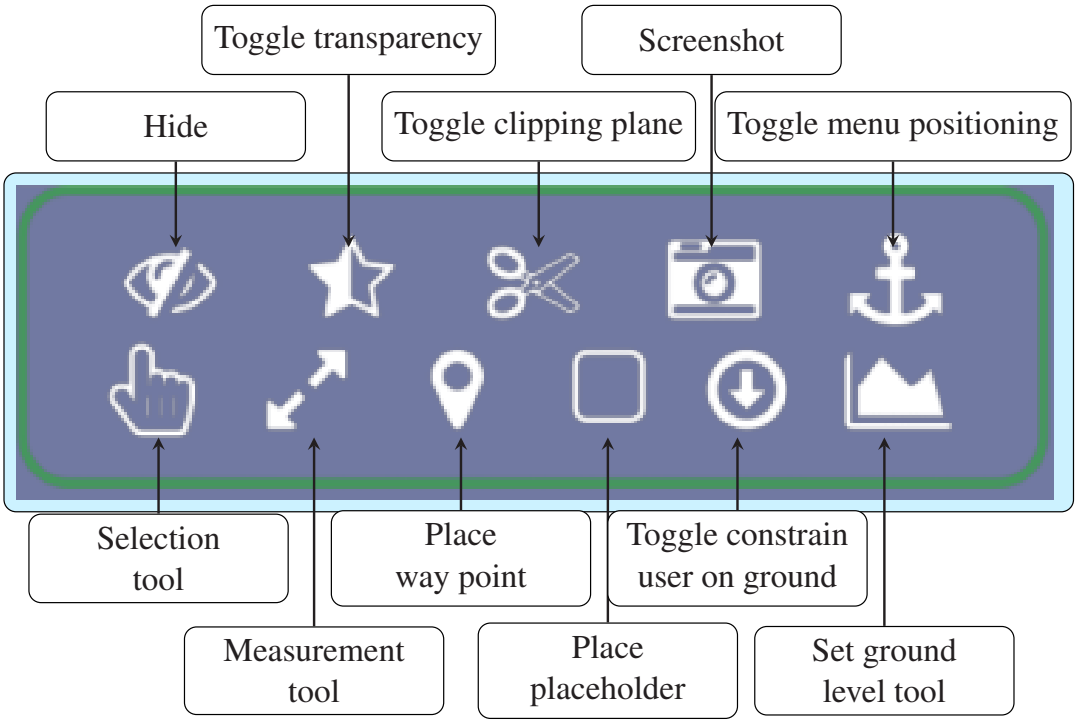


Fig. 6.20: First menu toolbar, tools management

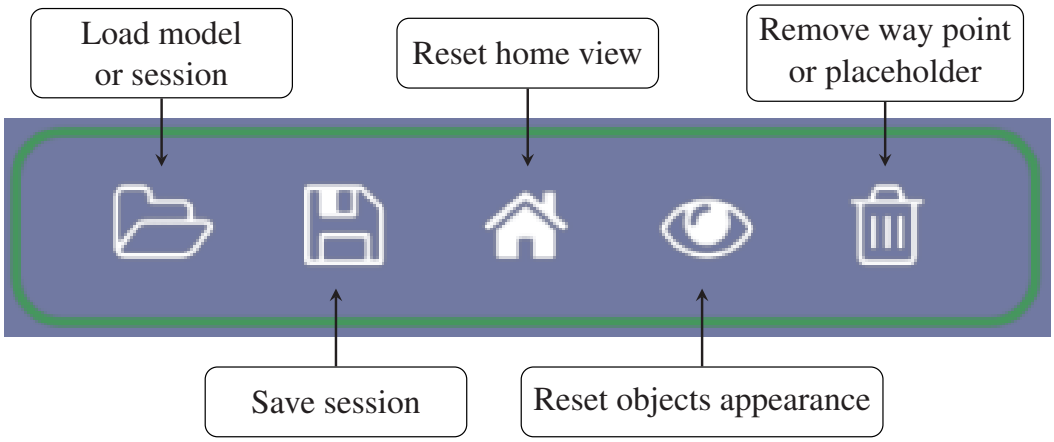


Fig. 6.21: Second menu toolbar, session management

The design review application was developed using PolyVR. The requirements to the design review system were the following:

- Aim at maximizing the acceptance of the system, especially for potential users
- Transfer the CAD geometry as accurately as possible
- Keep the product structure if possible
- Interact with the model by drag and drop
- Add tools to facilitate the exploration of the model
- Make the workflow to transfer CAD data into VR as simple and fast as possible
- Support common data exchange formats for the import in VR that are also available in most CAD systems as export formats.
- Deploy in a three sided active stereo CAVE system
- Support state of the art tracking from ART

Design Review Application

The design review application consists mainly of a viewer that loads a virtual model based on CAD data and allows the user to explore the model in detail and validate the construction progress. The main components are thus the import capabilities, the session management and the tools as well as navigation and interaction possibilities.
The user interface consists of a panel attached to the camera node, with three rows of tool buttons as well as a viewer for the product structure tree. The toolbars are depicted on fig. 6.20 and fig. 6.21
The basic interaction paradigm allows to either select objects, or drag and drop each part of the model to disassemble and reassemble. The parts do snap back to their original position.
The collaborative design review process is supported by tools, easing the exploration of the virtual model. The first set of tools can modify the appearance of parts, select parts (fig, 6.22), hide them, or set their appearance to semi-transparent.
The second set of tools directly help explore the model. The measurement tool is a set of three points in space, set by clicking on the surface of geometries of the model. Those three points are visually connected and the lengths of those three edges are displayed as overlay to the model as seen in fig. 6.23. In addition to the edge lengths, the three angles of that triangle are displayed, as well as the lengths of the normal projections of each point onto the opposing tangent. The clipping plane is a mathematical plane, visually clipping the rendered model to a visible half

space. The clipping plane can be dragged to easily position it in space. Depending on the current mode, the plane can be restricted to an axis of the world coordinate system, or handled freely without constraints.
The last two tools are elements that can be added to the scenes. The placeholder is a cube that can be resized using the handles on each cube side. It is used to represent components that are not yet designed, but where it might be helpful to see a representation of the space reserved for that component. The way point is an arrow that appears under the current position of the user. Creating such a way point stores the current camera position and orientation and associates the arrow with it. Clicking on the arrow instantly teleports the user to the stored position, like a bookmark in 3D space.

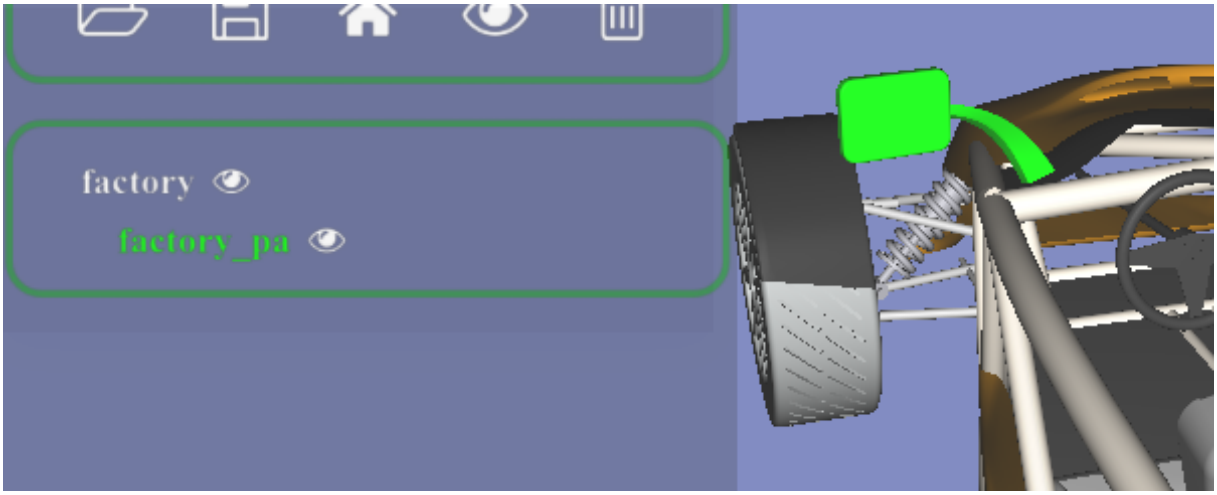


Fig. 6.22: The selection of an object will highlight the corresponding part in the CAD software, model [Sto04]

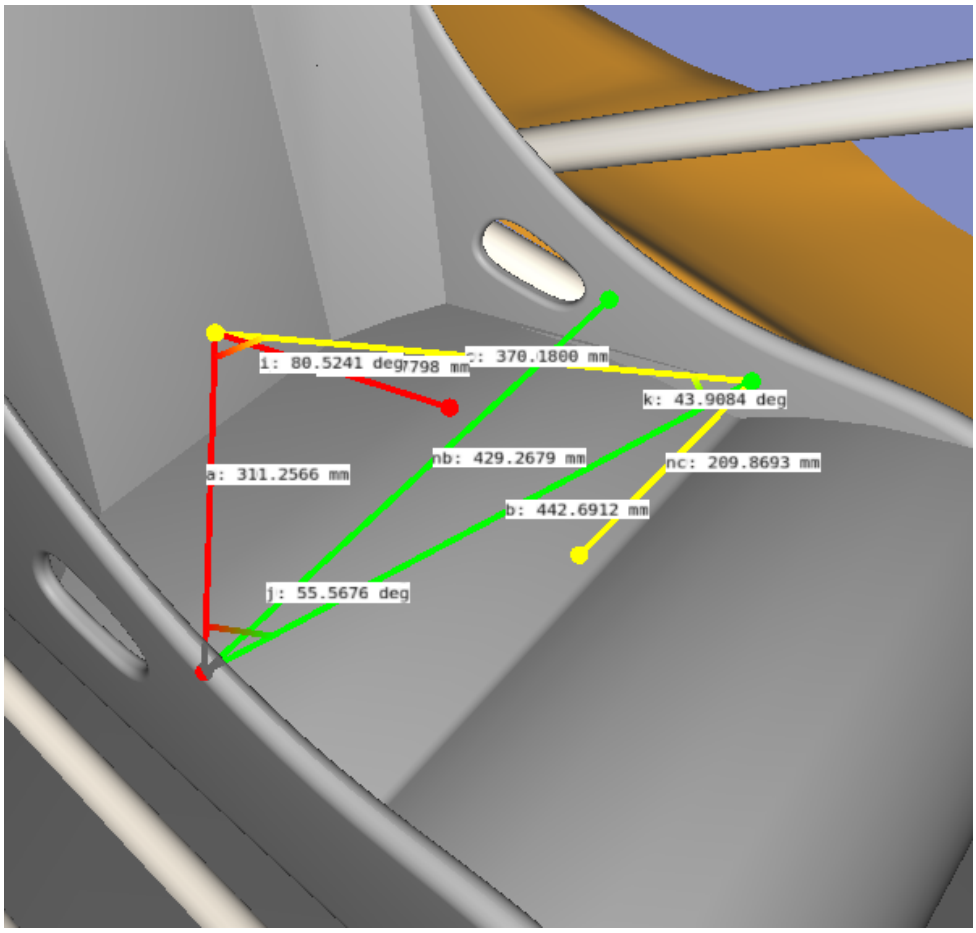
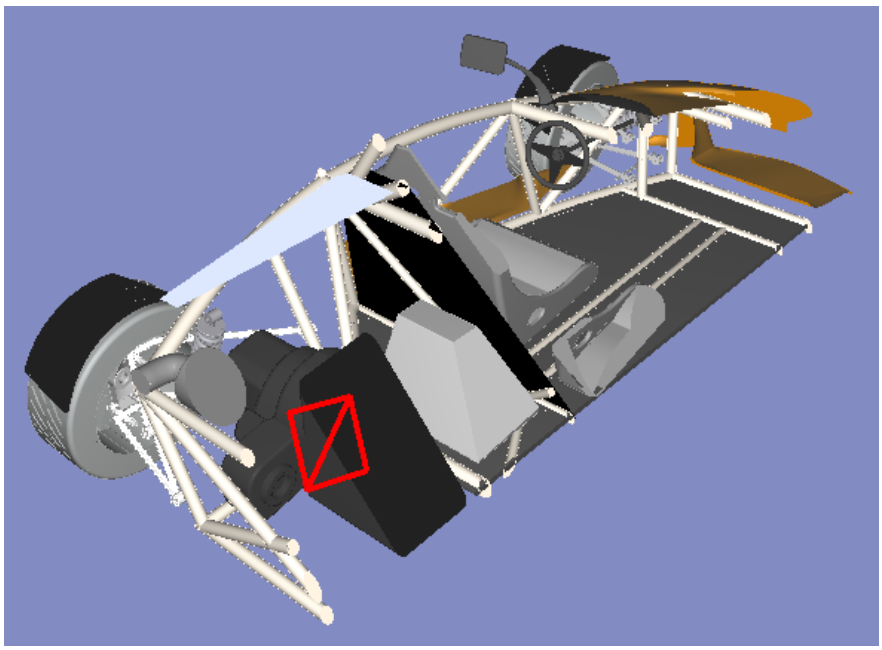
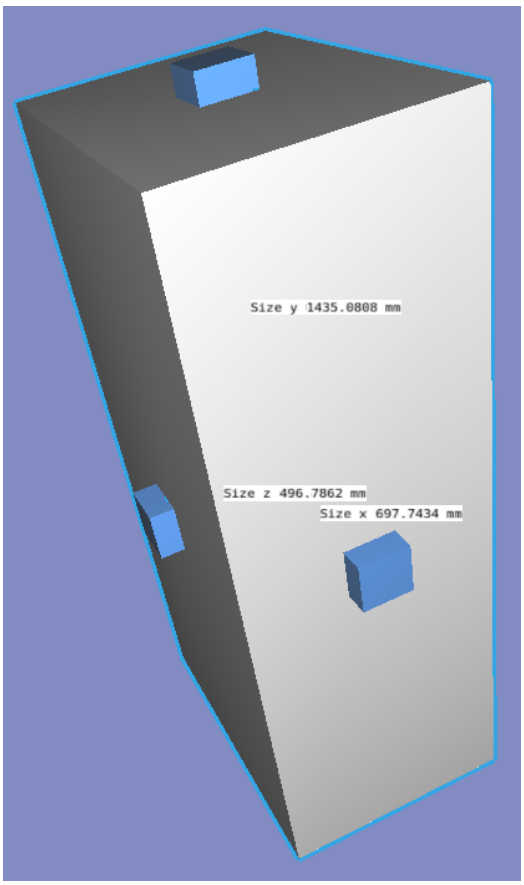


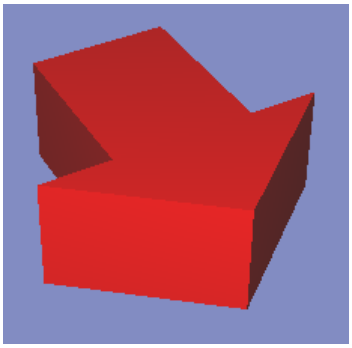
Fig. 6.23: Measuring tool, each click places a triangle corner, model [Sto04]



(a)

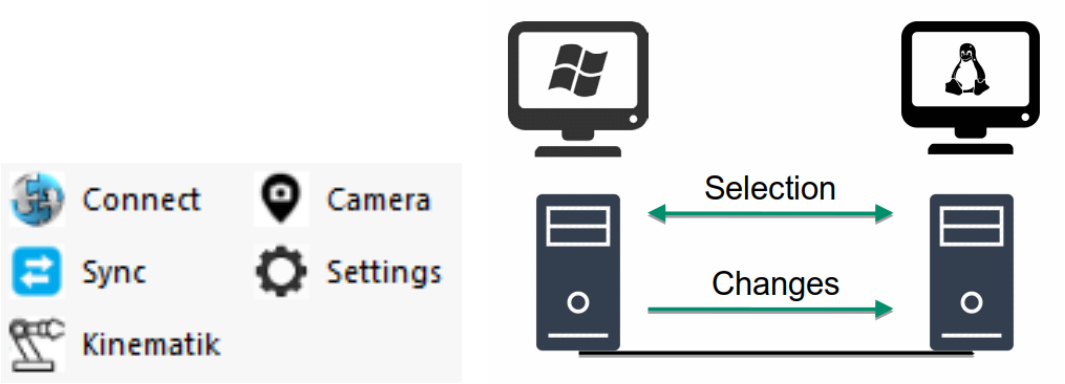


(b)



(c)

Fig. 6.24: b) Add placeholder to take missing components into account, a) Clipping plane, position and orientation in space with drag and drop, model [Sto04], c) Add way points to store camera position and orientation, click on arrow to navigate to that position



(a) (b)

Fig. 6.25: a) CAD plugin toolbar, b) CAD/VR bidirectional communication interface

CAD Plugin

The design review can use file import as primary virtualisation data source, but this means that the necessary files, MCAD STEP files for example, have to be exported from the CAD tool and stored on a network drive. This has the disadvantage that changes discussed during the design review, which are applied to the CAD model, cannot directly be applied to the virtual model without exporting the changes again. This becomes problematic when the CAD model grows in complexity and size, and the export gets slower and slower. The extra step to transfer the data greatly adds to the complexity of the virtualization workflow, and thus increases the risks that the process fails. The data export itself can fail as well as the import in PolyVR, and when the transfer succeeds it does usually come with data loss.

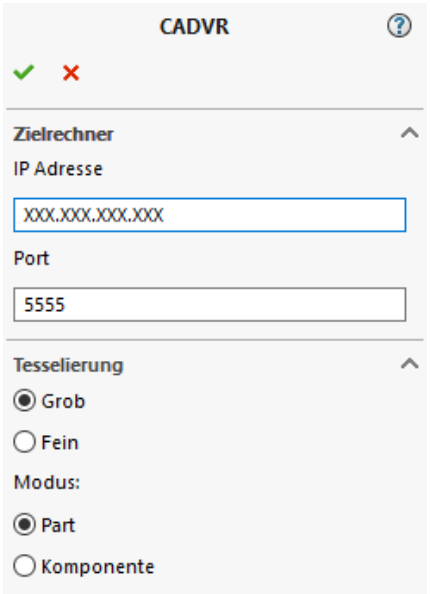


Fig. 6.26: CAD plugin settings

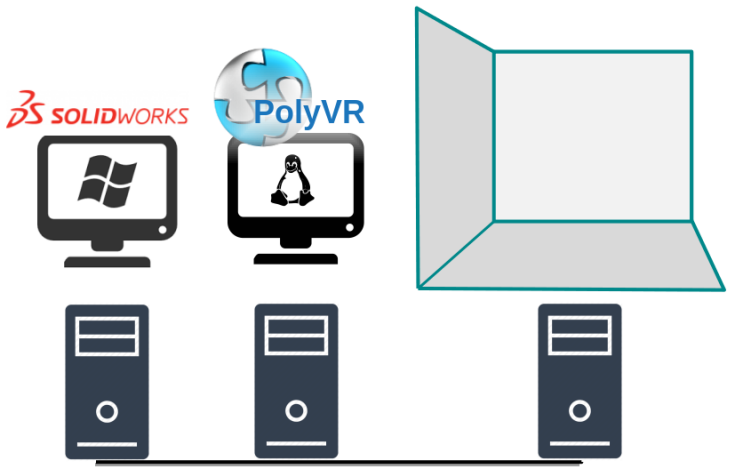


Fig. 6.27: Design review system overview, left the CAD workstation, right the CAVE visualisation system and in the middle the VR system with PolyVR

A way to improve this issue is to directly access the model information in the CAD system by using a plugin. This allows to send information directly from the CAD software to the virtual environment, even bidirectional. Such a plugin has been implemented for SolidWorks and Siemens NX. The use of the plugin drastically reduces the complexity and time to visualize the CAD model in VR. The construction engineer can press the sync button in the CAD software UI as depicted on fig. 6.25 a), and sees the 3D model appearing part by part in the immersive environment. It is still possible to explore the model as it builds up, while a loading bar indicates the virtualisation progress. Another major advantage lies in the possibility to automatically synchronize small changes. Changes to the CAD model can directly be applied to the CAD environment and will be directly sent to the VR system. This includes changes to part position and orientation as well as part geometries. Due to the bidirectional nature of the system, it is also possible to send information from the VR environment to the CAD software. This has been used to further extend the collaboration capabilities of the design review system by synchronizing the user selection in VR and CAD in both directions, as seen in figure 6.25 b).
The setup of the plugin was limited to parameterizing the communication interface. The necessary settings can be directly edited in a plugin dialog as seen in fig. 6.26.

CAVE Collaboration Space

The immersive hardware system installed at Schenck RoTec is a good example for an efficient and affordable solution, also for SMEs. The CAVE system has three projection surfaces, two side and one floor projection as depicted on fig. 6.27. The surfaces form a corner, creating a big view field. User can enter the setup and collaborate in this highly immersive space. One user can be tracked, his perspective is rendered in real-time to simulate a consistent space, allowing the virtual scene to be experienced as realistically as possible in one to one scale.

The tracking system also allows the use of a 6DoF interaction device, a so called Flystick. Figure 6.28 shows how the Flystick buttons are bound to basic functions like undo/redo or selection and manipulation of objects.

Design Review Conclusion and Outlook

The introduction of the design review using virtual engineering methods at Schenck RoTec has been very successful. The system is well established and regularly used, multiple times each month. The requirements of the company could all be satisfied.
The current version of the design review application only implements the MCAD interface. Even the kinematics transfer is only implemented as a prototype functionality. The next steps will lead to the integration of the virtualization workflow as described in chapter 6.2.3. This means the full kinematics information will have to be used in the virtualization, as well as the functional aspects like ECAD data and automation. The difficulty to do this right away is the need to further analyze the design process in SMEs, especially regarding the mapping of MCAD and ECAD components as well as the programming of the machines. This is important in order to generalize the approach and integrate it in the design review application as a sufficiently reliable virtualisation workflow.

6.3 Summary

In this section, the virtual reality authoring tool was validated regarding its capability to be employed for engineering applications. This was done by employing it for various projects, under the overarching theme of virtualization. The first domain was the content generation and management for open world systems. Those are key features of a virtual environment authoring

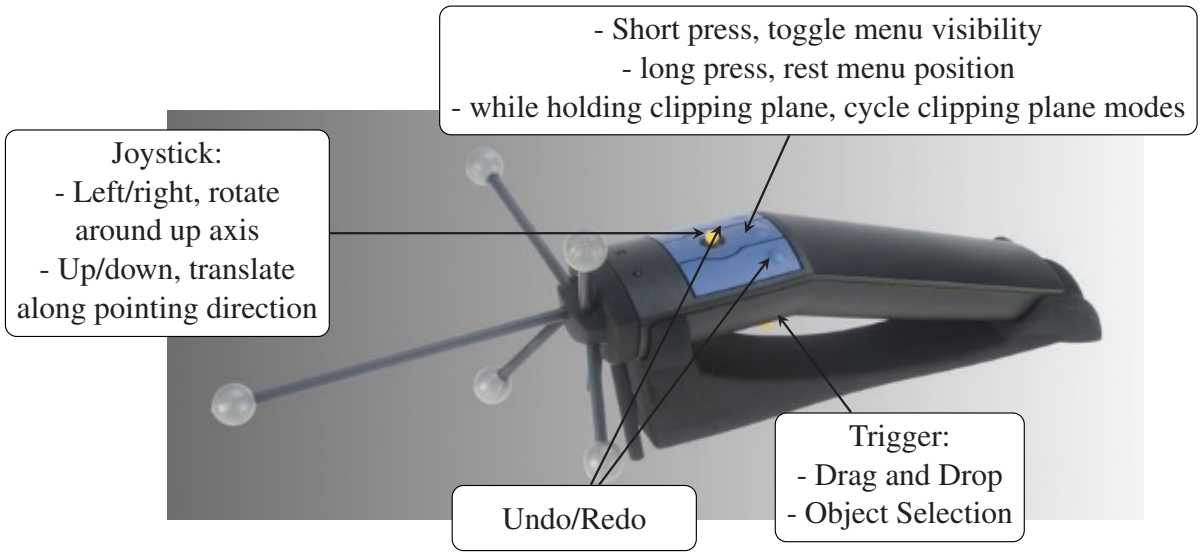


Fig. 6.28: Flystick key bindings, adapted from: [ART]

system. This includes handling efficiently large scale scenes with complex assets like roads, buildings and nature, as well as managing GIS data like raster topology data and planar map data. The second domain was a classical engineering application, the virtualization of machinery, especially complex production plants.
A topic that is addressed throughout all of the projects and application presented in this chapter is how to efficiently model intelligent behaviour. The use of ontologies and reasoning is shown for the various domains.
Validating the innovation of the authoring aspect of an authoring software is very challenging, as comparing software systems regarding usability, efficiency and intuitiveness is heavily dependent on the use case, user requirements and user skills or knowledge. To investigate this in depth goes beyond the scope of this thesis, but this chapter does provide some insights to specific aspects like the new possibilities due to the overall authoring workflow, or the possibilities offered by the key technology choices like the python scripting environment or the web based 2D UI design system.

7 Conclusion and Outlook

The previous chapters are reviewed and the proposed methods and systems are discussed. Then the research questions formulated in the introduction are discussed and answers are proposed. At last we discuss the role of virtual reality in the context of industry 4.0.

7.1 Summary

The aim of this thesis is to facilitate the introduction of virtual reality (VR) technologies into small and medium sized enterprises (SME) and specifically to introduce virtual engineering methods in the product development process. The main hurdle for SMEs are the high costs for introducing VR. On one hand, the hardware and software costs are very high, and operation and maintenance costs are even higher, because VR is not the core business of product developing companies. Furthermore, the currently available software heavily relies on VR expert handwork to build virtual interactive models, which is not practicable for the use in the product development process.
A VR framework is proposed in this work to address those problems. The tool is conceptually described in the methodology chapter 4. Then follows the implementation chapter and the validation chapter presents use cases in two domains, open world generation and driving simulation as well as the virtualization of integrated production facilities for the plant development process.

7.1.1 Methodology

This chapter draws an outline of a virtual reality authoring system that aims at greatly simplifying the virtualization workflow and the deployment of virtual reality applications in immersive hardware environments. Specialized modules are conceived to support the development of the various engineering applications. From the various common data interfaces to the use of ontologies for a semantic layer. A synergistic use of the various modules should allow the fast development of complex engineering applications. This system should enable the use of VR for SMEs, without the need to build up know-how in VR technologies, especially software development.

7.1.2 Implementation

PolyVR is a virtual reality framework designed to support the development of virtual environments for engineering applications. The core is built on the carefully selected open source libraries. It provides many interfaces for data exchange, file formats, communication interfaces and advanced hardware support. The spine of the software architecture is the scene graph library OpenSG, which excels in threading and clustering communication to allow for distributed visualisation applications. Advanced engineering modules for math, algorithms and data structures are the necessary tools to quickly build complex engineering applications. An intuitive UI and a scripting environment allow to easily develop application logic for virtual environments, with a low entry threshold making it ideal for student works and practical courses.
One of the most advanced modules is the reasoning system and the knowledge base module. They are key to further abstract application logic and can be employed to infuse virtual environments with a more flexible and generic intelligence.

7.1.3 Validation

The validation of the PolyVR system design has been achieved through the development of typical engineering applications in academic and industrial settings. The first application is a driving simulator with an open world generation based on GIS data. The second set of applications is in the domain of plant engineering, automating the virtualization of machinery in the product development process of integrated production lines.

7.2 Research Questions

Based on the validation, the research questions defined in the introduction can be discussed.

7.2.1 Virtual Environment Authoring

PolyVR aims at offering an intuitive UI to ease the authoring of virtual environments. On top of standard features like the scene graph viewer and the 3D view, there are some unique selling points. First there is the basic workflow, there is no edit and run mode. This avoids the need to restart the whole application to test it and allowing to dynamically edit the application logic. The second advantage of PolyVR is its use of flexible and versatile scripting languages. Python offers the greatest flexibility because it is possible to integrate external code and libraries into the VR application. JavaScript does the same for the design of 2D panels in the virtual environment. The last innovative feature regarding the authoring aspect in PolyVR is the modelling of ontologies, the connection to the scene graph and the live feedback from the reasoning system.

7.2.2 Virtual Engineering Applications

In contrast to gaming engines or authoring systems for games like Unity3D, PolyVR focuses on providing the necessary infrastructure for creating engineering applications. This starts with the numerous data interfaces and import formats to handle CAD data or ease the use of communication interfaces like OPCUA or ROS. Many math and algorithm modules provide an engineer with data models for space partitioning or geometry generation for data visualization.
The research and industry projects we worked on using PolyVR as main development tool also contributed each time with more specific modules, making the tool even more powerful. Especially due to the overall architecture and modularity of PolyVR, synergies between modules allow even faster and more flexible development.

7.2.3 Integrating Virtual Reality in Industry

PolyVR has been used for various industrial projects, but it has also been deployed at companies to implement virtual engineering workflows. This is one major breakthrough as it is a good basis for further developing the automated virtualization based on CAD data.

7.2.4 Adoption of Virtual Reality in Industry

Virtual Reality concepts have been around for decades. The field grew steadily but slowly with the advances in hardware technologies like computing capability and immersive devices. Nonetheless, it was hardly adopted by the industry, let alone could VR establish itself on the mass market. The reasons for this are definitely various and synergistic, but the main reason is the lack of added value compared to the amount of know-how and resources needed to employ virtual reality technologies. Currently available software solutions lacks the flexibility and automation capabilities for industrial usage of virtual reality, especially for deploying virtual engineering techniques in the product development process in small and middle sized enterprises. In the last years, this has begun to change. Virtual reality has found its way into society, which has found greater acceptance in the society. The mindset has changed from “do we really need this?” to “how can we make use of it?”. This new trend is paired with the ever growing complexity of product development, manufacturing and life cycle management. Companies have to deal with increasing complexity, the need for new technologies is growing steadily. More and more companies start discussing the integration of VR technologies and defining their needs in the context of their processes and workflows. This imposes new challenges for virtual reality software regarding interfaces, flexibility and performance.

7.2.5 Virtual Engineering Applications

Virtual engineering applications need to provide clear added value to engineering processes of an enterprise without the need for the company to build up VR know-how or apply big changes to their development workflows. Data and communication interfaces play a key role in integrating seamlessly VR in the processes and workflows of the enterprises as well as handling huge amounts of data while performing reasonably well. Once the virtualization process is established, the various engineering applications like design review, virtual commissioning, training or monitoring can be implemented using the generic modules of PolyVR. Industrial standards for data exchange like STEP or IFC allow to import CAD data, the geometric basis of the virtual mode. More advanced workflows can be set up via plugins for the native CAD software environments. Importing ECAD data and fusing it into the geometric MCAD model has been analysed, especially regarding the export capabilities of ECAD software. A prototype implementation does prove the feasibility of the overall virtualization concept, but more research is needed for the integration of ECAD data and the extension of the knowledge base to enable the functional execution of the wiring system. This applies as well to PLC programming logic. Much work is still needed to create a generic emulation system connected to the virtual environment.
Many modules in PolyVR allow to easily set up advanced functions such as generating geometry, visualizing metadata or interacting with paths or graph visuals in a virtual environment. These are the basic building blocks used for customizing virtual environments for various applications. The modular approach enables for a fast extension of the scripting bindings library, which in turn allows to quickly extend the available functions for the implementation of VR environments to fit the application requirements.

7.3 The Road Ahead

This work is a stepping stone towards fully automated generation of virtual environments for engineering applications. The first milestone shall be to achieve a fully automated virtualization for virtual commissioning. There is still much work to be done to achieve this goal. CAD data is very heterogeneous and difficult to fuse in order to create a single, consistent model. The knowledge base necessary for supporting the virtualization process will have to be further developed, growing domain by domain to reach a certain level of maturity. With each addition, the virtual models will be more dynamic and functional, integrating more and more of the available CAD information. Apart from integrating the CAD data, the integration of PLC programming is the next task to tackle. The best approach is still unclear, one possibility is to execute the programming in emulation software and develop an interface between the emulator and the virtual environment.
A further milestone is to add human models, necessary for many applications, like ergonomics, training and logistics. As it was the case with automation of the virtualization process, the challenge lies in avoiding the handwork needed to place and animate human models as well as script their behaviour. There are exchange formats to describe human behaviour as processes on different abstraction layers, even for very granular behaviour, where each gesture and motion is described in detail. This module will also have to integrate with the semantic layer modules to enable the simulation of intelligent agents.
A very important module is the tutoring system. A basic prototype has been implemented in the driving simulation described in the validation chapter 6.1. This system has to be further developed in order to further automate the tutoring process needed for advanced training applications. The basis for the automated tutoring is the semantic information contained in the training environment. The virtualization workflow is thus the key for a good training system. Last, there is the broad field of numerical simulations, necessary for example to simulate processing steps in production facilities. With the ever growing hardware capabilities, especially GPGPU resources, the computation of complex physical systems like fluids and flexible parts become feasible in real time, at least for reasonable sized simulation models. The use of CUDA accelerated Krylov subspace methods are highly promising to run typical engineering simulations like multi body dynamics, finite element methods or computational fluid dynamics as interactive simulations that can be manipulated in real time in virtual environments.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[AB07] ABRAMOVICI, Michael ; BELLALOUNA, Fahmi: Integration and complexity management within the mechatronics product development. In: *Advances in Life Cycle Engineering for Sustainable Manufacturing Businesses*. Springer, London, 2007, S. 113–118

[AB08] ABRAMOVICI, M ; BELLALOUNA, F: Service oriented architecture for the integration of domain-specific PLM systems within the mechatronic product development. In: *Proceedings of the 7th International Symposium on Tools and Methods of Competitive Engineering (TMCE 2008), Izmir, Turkey*, 2008, S. 941–953

[AF16] ALLIEZ, Pierre ; FABRI, Andreas: CGAL: the computational geometry algorithms library. In: *ACM SIGGRAPH 2016 Courses* ACM, 2016, S. 8

[al.04] AL., Daniel S.: *Libssh2 library*. `https://www.libssh2.org/`, 2004. – Accessed: 2018-01-24

[al.13] AL., Alexander R.: *FreeOPCUA library*. `https://github.com/FreeOpcUa`, 2013. – Accessed: 2018-01-24

[ART] ART: *ART - Advanced Realtime Tracking GmbH*. `https://ar-tracking.com/`, . – Accessed: 2019-04-14

[ASO14] ALEKSANDROV, Kiril ; SCHUBERT, Viktor ; OVTCHAROVA, Jivka: Skill-Based Asset Management: A PLM-Approach for Reconfigurable Production Systems. In: *IFIP International Conference on Product Lifecycle Management* Springer, 2014, S. 465–474

[BBA+10] BURGER, Alexander ; BITTEL, Vitalis ; AWAD, Ramez ; STANEV, Stilian ; OVTCHAROVA, Jivka: Customer-Orientation Using Integration and Individualization Aspects Enabling the Transition from Manufacturer to Solution Provider. In: *3rd International Workshop Ïnnovation in Information Technologies: Theory and Practice". Dresden, Germany*, 2010

[BBAO11] BURGER, Alexander ; BITTEL, Vitalis ; AWAD, Ramez ; OVTCHAROVA, Jivka: Design for customer-Sustainable customer integrations into the development

processes of product-service system providers. In: *Proceedings of MECIT international conference on applied information and communications technology, Muscat*, 2011, S. 22–23

[BC03] BURDEA, Grigore C. ; COIFFET, Philippe: *Virtual reality technology*. John Wiley & Sons, New Jersey, USA, 2003

[Bel19] BELLALOUNA, F: VR-Based Design Process of Industrial Products. In: *Proceedings of the 7th International Conference on Competitive Manufacturing (COMA 2019), Stellenbosch, South Africa*, 2019, S. 240–245

[Ben19] BENEDIX, Anne-Christine: *Virtualisierung der Kinematik von CAD-Modellen im Anlagenbau*, Karlsruhe Institute of Technology, Institute for Information Management in Engineering, Bachelorthesis, 2019

[BLHL01] BERNERS-LEE, Tim ; HENDLER, James ; LASSILA, Ora: THE SEMANTIC WEB. In: *Scientific American* 284 (2001), Nr. 5, 34–43. `http://www.jstor.org/stable/26059207`. – ISSN 00368733, 19467087

[BMKSH09a] BACHVAROV, Angel ; MALESHKOV, Stoyan ; KATICIC, Jurica ; STOYANOVA (HÄFNER), Polina: Design-by-the-Customer through Virtual Reality. In: *4th International Conference on Advanced Research in Virtual and Rapid Prototyping VRAP 2009, Leiria, Portugal*, 2009

[BMKSH09b] BACHVAROV, Angel ; MALESHKOV, Stoyan ; KATICIC, Jurica ; STOYANOVA (HÄFNER), Polina: Product Individualization by the Customer through Virtual Reality Support. In: *Proceedings of the 5-th International Conference on Mass Customization & Personalization MCPC 2009, Helsinki, Finland*, 2009

[BO13] BURGER, Alexander ; OVTCHAROVA, Jivka: Design for customer-sustainable customer integration based upon a customer-driven solution configurator. In: *The Philosopher's Stone for Sustainability*. Springer, Berlin, Heidelberg, 2013, S. 263–268

[BVHO15] BAYART, Benjamin ; VARTANIAN, Alexis ; HAEFNER, Polina ; OVTCHAROVA, Jivka: TechViz XL helps KITs Formula Student car "become alive". In: *Virtual Reality (VR), 2015 IEEE* IEEE, 2015, S. 395–396

[CCKS05] CHANG, Paul Hsueh-Min ; CHIEN, Yu-Hung ; KAO, Edward Chao-Chun ; SOO, Von-Wun: A knowledge-based scenario framework to support intelligent planning characters. In: *International Workshop on Intelligent Virtual Agents* Springer, Berlin, Heidelberg, 2005, S. 134–145

[Cou13] COUMANS, Erwin: Bullet physics library. In: *Open source: bulletphysics. org* 15 (2013), Nr. 49, S. 5

[Cov] COVISE: *Covise*. https://www.hlrs.de/covise, . – Accessed: 2019-04-14

[DHO15] DÜCKER, Jana ; HÄFNER, Polina ; OVTCHAROVA, Jivka: Wirtschaftlichkeit von Virtual Reality für den Mittelstand. In: *VAR^2 2015, 3. Fachkonferenz zu VR/AR-Technologien in Anwendung und Forschung in Chemnitz*, 2015

[DHO16a] DÜCKER, Jana ; HÄFNER, Polina ; OVTCHAROVA, Jivka: Methodology for Efficiency Analysis of VR Environments for Industrial Applications. In: *International Conference on Augmented Reality, Virtual Reality and Computer Graphics* Springer, Cham, 2016, S. 72–88

[DHO16b] DÜCKER, Jana ; HÄFNER, Polina ; OVTCHAROVA, Jivka: Virtual Reality im Alltag - Erfahrungswerte von VR-Industrieanwender in Bezug auf Einsatz, Probleme und Wirtschaftlichkeit. In: *INNTERACT CONFERENCE. 3D Sensation*, 2016

[DV11] DIMITROV, Lubomir ; VALCHKOVA, Fani: Problems with 3D Data Exchange Between CAD Systems Using Neutral Formats. In: *Proceedings in Manufacturing Systems 6.3*, 2011

[EFJS15] EMMER, Christian ; FRÖHLICH, Arnulf ; JÄKEL, Volker ; STJEPANDIĆ, Josip: Advances in standardized approach to ECAD/MCAD collaboration. In: *Journal of Aerospace Operations* 3 (2015), Nr. 3, 4, S. 185–201

[EHH17] ELSTERMANN, Matthes ; HUSEN, Christian v. ; HÄFNER, Polina: The dimenSion concept-approaching service prototyping from a multi-aspect description perspective. In: *Proceedings of the QUIS15 International Research Symposium on Service Excellence in Management, June 12-15 2017, University of Porto, Porto, Portugal* (2017)

[EHOO11] EICHHORN, Daniel ; HERTER, Johannes ; OBERWEIS, Andreas ; OVTCHAROVA, Jivka: An Approach for a Domain-spanning Collaboration Platform for Decision Support Using Immersive Visualization Techniques in Product Manufacturing. In: *1st International Workshop on Collaborative Usage and Development of Models and Visualizations, CollabViz*, 2011

[EO14a] ELSTERMANN, Matthes ; OVTCHAROVA, Jivka: Abstract Layers in PASS–A Concept Draft. In: *International Conference on Subject-Oriented Business Process Management* Springer, 2014, S. 125–136

[EO14b] ELSTERMANN, Matthes ; OVTCHAROVA, Jivka: An Editing Concept for PASS Layers. In: *International Conference on Subject-Oriented Business Process Management* Springer, Cham, 2014, S. 137–146

[FK00] FARR, Tom G. ; KOBRICK, Mike: The shuttle radar topography mission / Jet Propulsion Laboratory CA, USA. 2000. – Forschungsbericht

[Fri15] FRIEBE, Sebastian: *Verkehrssimulation*, Karlsruhe Institute of Technology, Institute for Information Management in Engineering, Bachelorthesis, 2015

[GBL06] GRIMALDO, Francisco ; BARBER, Fernando ; LOZANO, Miguel: An ontology-based approach for IVE+ VA. In: *IVEVA International Conference*, 2006

[Git] GITHUB: *Github Octoverse*. `https://octoverse.github.com/projects#languages`, . – Accessed: 2019-04-14

[GLP76] GELENBE, Erol ; LABETOULLE, Jacques ; PUJOLLE, Guy: Performance evaluation of the HDLC protocol. In: *Computer Networks (1976)* 2 (1976), Nr. 4-5, S. 409–415

[GOG07] GEORGIEV, Iliya ; OVTCHAROVA, Jivka ; GEORGIEV, Ivo: Modelling web services for PLM distributed system. In: *International Journal of Product Lifecycle Management* 2 (2007), Nr. 1, S. 30–49

[Gor12] GORTLER, Steven J.: *Foundations of 3D computer graphics*. Cambridge, Mass. : MIT Press, 2012. – ISBN 978–0–262–01735–0; 0–262–01735–0

[GPC+07] GEORGOULIAS, Konstantinos ; PAPAKOSTAS, Nikos ; CHRYSSOLOURIS, George ; OVTCHAROVA, Jivka ; KRAPPE, Hardy ; STANEV, Stilian: Flexibility assessment platform for the factory of the future. In: *Technology Management Conference (ICE), 2007 IEEE International* IEEE, 2007, S. 1–8

[Gre09] GREENBLATT, Marshall: *MS Windows NT Kernel Description*. `https://bitbucket.org/chromiumembedded/cef`, 2009. – Accessed: 2018-01-19

[Häf17a] HÄFNER, Victor: Modelling Smart Virtual Environments. In: *4. Fachkonferenz zu VR/AR-Technologien in Anwendung und Forschung an der Professur Werkzeugmaschinen und Umformtechnik, 2017, Chemnitz, Germany, ISBN 978-3-00-058419-0, Pages 151-162* (2017)

[Häf17b] HÄFNER, Victor: PolyVR - A Virtual Reality Authoring System. In: *EuroVR 2014, Bremen, Germany* (2017)

[Ham10] HAMMEL, Michael J.: Mongoose: an embeddable web server in C. In: *Linux Journal* 2010 (2010), Nr. 192, S. 2

[HBO13] HERTER, Johannes ; BROWN, Ross ; OVTCHAROVA, Jivka: A visual language for the collaborative visualization of integrated conceptual models in product development scenarios. In: *Smart Product Engineering*. Springer, Berlin, Heidelberg, 2013, S. 805–814

[HDO15] HÄFNER, Polina ; DÜCKER, Jana ; OVTCHAROVA, Jivka: Virtual Reality für den Mittelstand. In: *18. IFF-Wissenschaftstage in Magdeburg 2015, Digital Engineering zum Planen, Testen und Betreiben technischer Systeme*, 2015

[HEHO16] HÄFNER, Polina ; ELSTERMANN, Matthes ; HÄFNER, Victor ; OVTCHAROVA, Jivka: Virtual Reality Based Rapid Service Prototyping Tool. In: *Proceedings of EuroVR Conference 2016, Athene, Greece* (2016)

[HH+18] HAINES, Eric ; HOFFMAN, Naty u. a.: *Real-time rendering*. CRC Press, Florida, USA, 2018

[HHHO14] HUMMEL, Simon ; HÄFNER, Victor ; HÄFNER, Polina ; OVTCHAROVA, Jivka: New Techniques for Hand Pose Estimation Based on Kinect Depth Data. In: *Proc. of Conference and Exhibition of the European Association of Virtual and Augmented Reality'14*, 2014

[HHO13] HÄFNER, Polina ; HÄFNER, Victor ; OVTCHAROVA, Jivka: Teaching methodology for virtual reality practical course in engineering education. In: *Procedia Computer Science* 25 (2013), S. 251–260

[HHO14] HÄFNER, Polina ; HÄFNER, Victor ; OVTCHAROVA, Jivka: Experiencing physical and technical phenomena in schools using virtual reality driving simulator. In: *International Conference on Learning and Collaboration Technologies* Springer, Cham, 2014, S. 50–61

[HHWO13] HÄFNER, Polina ; HÄFNER, Victor ; WICAKSONO, Hendro ; OVTCHAROVA, Jivka: Semi-automated Ontology Population from Building Construction Drawings. In: *5th International Conference on Knowledge Engineering and Ontology Development (KEOD)*, 2013, S. 379–386

[HO13] HOPF, Michael ; OVTCHAROVA, Jivka: Integration of virtualized environments in PDM systems for embedded software product development. In: *Procedia CIRP* 11 (2013), S. 346–351

[HO16] HERTER, Johannes ; OVTCHAROVA, Jivka: A model based visualization framework for cross discipline collaboration in Industry 4.0 scenarios. In: *Procedia CIRP* 57 (2016), S. 398–403

[Hri19] HRISTOV, Anton M.: *Virtualisierung eines Touch Panels zur Programmierung einer virtuellen Anlage*, Karlsruhe Institute of Technology, Institute for Information Management in Engineering, Bachelorthesis, 2019

[HSD+14] HÄFNER, Polina ; SEESSLE, Julia ; DÜCKER, Jana ; ZIENTEK, Matthias ; SZELIGA, Filip: Interactive Visualization of Energy Effciency Concepts Using Virtual Reality. In: *EuroVR 2014, Bremen, Germany*, 2014

[HSG+13] HANDFEST, Alexander ; SCHRÖDER, Michael ; GREFE, Philip ; CARRA, Pablo ; HÄFNER, Polina: Integration eines realen Fahrzeugs in eine Mixed-Reality-Fahrsimulation. In: *Wissenschaftliche Konferenz "Wirtschaft und Technologie im Dienst der Gesellschaft"2013, Sofia, Bulgarien*, 2013

[HSHR13] HÄFNER, Victor ; SIEBEL, Jan ; HÄFNER, Polina ; ROGALSKI, Sven: Interaktive Flexibilitätsbewertung in virtuellen Welten. In: *Zeitschrift für wirtschaftlichen Fabrikbetrieb (ZWF), 108. Jahrgang, Hanser Verlag, Heft März 2013, S. 123-127*, 2013, S. 123–127

[HSO08] HERTER, Johannes ; SCHOTTE, Wolfgang ; OVTCHAROVA, Jivka: Bridging vr and item-based plm systems using an object oriented approach. In: *PLM08-5th International Conference on Product Lifecycle Management, Seoul, South Korea*, 2008

[HSW+14] HÄFNER, V ; SCHINDLER, J ; WEIK, N ; MAYER, T ; BALAKRISHNAN, S ; NARAYANAN, R ; BERA, S ; EVERS, F: Density of states in graphene with vacancies: midgap power law and frozen multifractality. In: *Physical review letters* 113 (2014), Nr. 18, S. 186802

[HTC] HTC: *HTC Vive Headset*. `https://www.vive.com/`, . – Accessed: 2019-04-14

[HVDF+14] HUGHES, John F. ; VAN DAM, Andries ; FOLEY, James D. ; MCGUIRE, Morgan ; FEINER, Steven K. ; SKLAR, David F. ; AKELEY, Kurt: *Computer graphics: principles and practice*. Addison-Wesley Professional, Boston, USA, 2014

[HVH+13] HÄFNER, Polina ; VINKE, Christina ; HÄFNER, Victor ; OVTCHAROVA, Jivka ; SCHOTTE, Wolfgang: The impact of motion in virtual environments on memorization performance. In: *Computational Intelligence and Virtual Environments for Measurement Systems and Applications (CIVEMSA), 2013 IEEE International Conference on* IEEE, 2013, S. 104–109

[HW08] HAKLAY, Mordechai ; WEBER, Patrick: Openstreetmap: User-generated street maps. In: *IEEE Pervasive Computing* 7 (2008), Nr. 4, S. 12–18

[ICI] ICIDO: *ICIDO*. `https://virtualreality.esi-group.com`, . – Accessed: 2019-04-14

[Int] INTERACTIVE, Paradox: *Paradox Interactive Cities-Skylines Trailer*. `https://www.paradoxplaza.com/cities-skylines/`, . – Accessed: 2019-04-14

[KHO13] KATICIC, Jurica ; HÄFNER, Polina ; OVTCHAROVA, Jivka: Methodology for immersive emotional assessment of virtual product design by customers. In:

Proceedings of the 5th Joint Virtual Reality Conference Eurographics Association, 2013, S. 77–82

[KHO15] KATICIC, Jurica ; HÄFNER, Polina ; OVTCHAROVA, Jivka: Methodology for emotional assessment of product design by customers in virtual reality. In: *Presence: Teleoperators and Virtual Environments* 24 (2015), Nr. 1, S. 62–73

[KKB+17] KIESEL, Markus ; KLIMANT, Philipp ; BEISHEIM, Nicolai ; RUDOLPH, Stephan ; PUTZ, Matthias: Using Graph-based Design Languages to Enhance the Creation of Virtual Commissioning Models. In: *Procedia CIRP* 60 (2017), 12, S. 279–283. http://dx.doi.org/10.1016/j.procir.2017.01.047. – DOI 10.1016/j.procir.2017.01.047

[KSO07a] KRAPPE, Hardy ; STANEV, Stilian ; OVTCHAROVA, Jivka: Integration of flexible manufacturing and change management processes in a service-oriented architecture. In: *Innovations and Advanced Techniques in Computer and Information Sciences and Engineering*. Springer, Dordrecht, 2007, S. 213–218

[KSO+07b] KRAPPE, Hardy ; STANEV, Stilian ; OVTCHAROVA, Jivka ; GEORGOULIAS, Konstantinos ; CHRYSSOLOURIS, George ; OLA, Hischam A.: Development of Flexibility Methods and their Integration into Change Management Processes for Agile Manufacturing. In: *New technologies for the intelligent design and operation of manufacturing networks, Fraunhofer IRB Verlag Stuttgart* (2007), S. 37–52

[Lab] LABS, Thalmic: *Thalmic Labs Myo Armband*. https://xinreality.com/wiki/Myo, . – Accessed: 2019-04-14

[LB17] LORENZ, Mario ; BERNHAGEN, Max: Chancen und Risiken von VR in der Produktentwicklung. In: *VAR^2 2017, 4. Fachkonferenz zu VR/AR-Technologien in Anwendung und Forschung in Chemnitz*, 2017, S. 0

[LBD+18] LORENZ, Mario ; BRADE, Jennifer ; DIAMOND, Lisa ; SJÖLIE, Daniel ; BUSCH, Marc ; TSCHELIGI, Manfred ; KLIMANT, Philipp ; HEYDE, Christoph-E ; HAMMER, Niels: Presence and User Experience in a Virtual Environment under the Influence of Ethanol: An Explorative Study. In: *Scientific reports* 8 (2018), Nr. 1, S. 6407

[LGMP16] LUPINETTI, Katia ; GIANNINI, Franca ; MONTI, Marina ; PERNOT, Jean-Philippe: Automatic extraction of assembly component relationships for assembly model retrieval. In: *Procedia CIRP* 50 (2016), S. 472–477

[LGMP17] LUPINETTI, Katia ; GIANNINI, Franca ; MONTI, Marina ; PERNOT, Jean-Philippe: Identification of functional components in mechanical assemblies. In: *Procedia CIRP* 60 (2017), S. 542–547

[LM06] LEITNER, Stefan-Helmut ; MAHNKE, Wolfgang: OPC UA–service-oriented architecture for industrial applications. In: *ABB Corporate Research Center* (2006)

[LO13] LINTALA, Marja ; OVTCHAROVA, Jivka: Enhancing system lifecycle processes by integrating functional safety information from practice into design requirements. In: *International Journal of Advanced Robotic Systems* 10 (2013), Nr. 10, S. 376

[LRP+15] LORENZ, Mario ; RIEDEL, Tino ; PÜRZEL, Franziska ; WITTSTOCK, Volker ; HOFFMANN, Alexander ; SPRANGER, Michael: An Automated Way for Conversion from CAD to Virtual Reality. In: *Konstruktion* 1/2 (2015), 01, S. 62–66

[LSPV15] LAWONN, Kai ; SMIT, Noeska N. ; PREIM, Bernhard ; VILANOVA, Anna: Illustrative Multi-volume Rendering for PET/CT Scans. In: *VCBM*, 2015

[LSR+16] LORENZ, Mario ; SPRANGER, Michael ; RIEDEL, Tino ; PÜRZEL, Franziska ; WITTSTOCK, Volker ; KLIMANT, Philipp: CAD to VR–a methodology for the automated conversion of kinematic CAD models to virtual reality. In: *Procedia Cirp* 41 (2016), S. 358–363

[LW85] LEVOY, Marc ; WHITTED, Turner: *The use of points as a display primitive*. Citeseer, University of North Carolina, Department of Computer Science, 1985

[MBC+12] MALESHKOV, Stoyan ; BACHVAROV, Angel ; CHOTROV, Dimo ; OVTCHAROVA, Jivka ; KATICIC, Jurica: Using implicit features for enhancing the immersive object representation in multimodal virtual reality environments. In: *Virtual Environments Human-Computer Interfaces and Measurement Systems (VECIMS), 2012 IEEE International Conference on* IEEE, 2012, S. 91–96

[McK47] MCKINLEY, Robert W.: *IES lighting handbook: The standard lighting guide*. McGraw-Hill, New York, USA, 1947

[MCSG15] MESSAOUD, Mezati ; CHERIF, Foudil ; SANZA, Cédric ; GAILDRAT, Véronique: An ontology for semantic modelling of virtual world. In: *International Journal of Artificial Intelligence & Applications* 6 (2015), Nr. 1, S. 65

[Mica] MICROSOFT: *Microsoft Hololens Headset*. `https://www.microsoft.com/en-us/hololens/`, . – Accessed: 2019-04-14

[Micb] MICROSOFT: *Microsoft Kinect*. `https://developer.microsoft.com/en-us/windows/kinect`, . – Accessed: 2019-04-14

[Mic19] MICHELS, Felix: *Echtzeitfähige Verkehrssimulation für einen Fahrsimulator in virtueller Realität*, Karlsruhe Institute of Technology, Institute for Information Management in Engineering, Masterthesis, 2019

[Mot] MOTION, Leap: *Leap Motion device*. `https://www.leapmotion.com/`, . – Accessed: 2019-04-14

[MR13] MEYEROVICH, Leo A. ; RABKIN, Ariel S.: Empirical analysis of programming language adoption. In: *ACM SIGPLAN Notices* Bd. 48 ACM, 2013, S. 1–18

[MSO07] MAHL, Alexander ; SEMENENKO, Anatoli ; OVTCHAROVA, Jivka: Virtual Organisation In Cross Domain Engineering. In: *Establishing the Foundation of Collaborative Networks*. Springer, Boston, MA, 2007, S. 601–608

[NBO13] NIKNAM, Masoud ; BONNAL, Pierre ; OVTCHAROVA, Jivka: Configuration management maturity in scientific facilities. In: *International Journal of Advanced Robotic Systems* 10 (2013), Nr. 12, S. 404

[NO13] NIKNAM, Masoud ; OVTCHAROVA, Jivka: Towards Higher Configuration Management Maturity. In: *IFIP International Conference on Product Lifecycle Management* Springer, Berlin, Heidelberg, 2013, S. 396–405

[OHH+15] OVTCHAROVA, Jivka ; HÄFNER, Polina ; HÄFNER, Victor ; KATICIC, Jurica ; VINKE, Christina: Innovation braucht resourceful humans, Aufbruch in eine neue Arbeitskultur durch virtual engineering. In: *Zukunft der Arbeit in Industrie 4.0*. Springer, Berlin, Heidelberg, 2015, S. 111–124

[OMG+11] OVTCHAROVA, Jivka ; MARINOV, Milan ; GUTU, Dan ; SZOTS, András Simonyi u. a.: Representation of cross-domain design knowledge through ontology based functional models. In: *DS 68-6: Proceedings of the 18th International Conference on Engineering Design (ICED 11), Impacting Society through Engineering Design, Vol. 6: Design Information and Knowledge, Lyngby/Copenhagen, Denmark, 15.-19.08. 2011*, 2011

[OMK06] OVTCHAROVA, Jivka ; MAHL, Alexander ; KRIKLER, Robert: Approach for a Rule Based System for Capturing and Usage of Knowledge in the Manufacturing Industry. In: *Knowledge Enterprise: Intelligent Strategies In Product Design, Manufacturing, and Management*. Springer, Boston, MA, 2006, S. 134–143

[ORK+08] OLA, Hischam A. ; ROGALSKI, Sven ; KRAHTOV, Konstantin ; STANEV, Stilian ; KRAPPE, Hardy ; OVTCHAROVA, Jivka: Efficient change management for the flexible production of the future. In: *Management* (2008), S. 1–15

[Ovt10] OVTCHAROVA, JG: Virtual engineering: principles, methods and applications. In: *DS 60: Proceedings of DESIGN 2010, the 11th International Design Conference, Dubrovnik, Croatia*, 2010

[Ovt15] OVTCHAROVA, Jivka: Virtuelles Abbild - neue Ingenieurmethoden für Industrie 4.0. In: *VAR^2 2015, 3. Fachkonferenz zu VR/AR-Technologien in Anwendung und Forschung in Chemnitz*, 2015

[Pol15] POLLARI, Greg: ECAD & MCAD Model, Virtual Integration Using Data Interoperability Standards. In: *Global Product Data Interoperability Summit*, 2015

[Rei02] REINERS, Dirk: *OpenSG: A scene graph system for flexible and efficient realtime rendering for virtual and augmented reality applications*, Technische Universität Darmstadt, Diss., 2002

[RO09] ROGALSKI, Sven ; OVTCHAROVA, Jivka: Flexibilitätsbewertung von Produktionssystemen: ecoFLEX-eine branchenübergreifende Methodik. In: *ZWF Zeitschrift für wirtschaftlichen Fabrikbetrieb* 104 (2009), Nr. 1-2, S. 64–70

[RoT16] ROTEC, Schenck: *Schenck RoTec Designreview*. `https://schenck-rotec.de/unternehmen/news/detail/cave-visionen-zum-leben-erwecken.html`, 2016. – Accessed: 2018-01-27

[RWO12] ROGALSKI, Sven ; WICAKSONO, Hendro ; OVTCHAROVA, Jivka: Resource-Efficient Planning in Production through Flexibility Measurements. In: *10th International Conference on Manufacturing Research (ICMR 2012), Birmingham, UK*, 2012

[SC18] SHERMAN, William R. ; CRAIG, Alan B.: *Understanding virtual reality: Interface, application, and design*. Morgan Kaufmann, Burlington, MA, USA, 2018

[Sch19] SCHRÖDER, Michael: *Simulator Hardware Interface*. `https://michi.bayern`, 2019. – Accessed: 2019-01-24

[SHO14] SCHUCK, Hendrik ; HÄFNER, Victor ; OVTCHAROVA, Jivka: Qualifizierung eines neuartigen Vibrationsverfahrens zur Dekontamination von Rohrleitungen. In: *Digital Engineering zum Planen, Testen und Betreiben technischer Systeme, 17. IFF-Wissenschaftstage, Fraunhofer - Institut für Fabrikbetrieb und -automatisierung IFF, Magdeburg*, 2014

[Sie] SIEMENS: *Siemens PLC S7*. `https://new.siemens.com/global/en/products/automation/systems/industrial/plc.html`, . – Accessed: 2019-04-14

[SKAO+08] STANEV, Stilian ; KRAPPE, Hardy ; ABUL OLA, Hischam ; GEORGOULIAS, Konstantinos ; PAPAKOSTAS, Nikolaos ; CHRYSSOLOURIS, George ; OVTCHAROVA, Jivka: Efficient change management for the flexible production of the

future. In: *Journal of Manufacturing Technology Management* 19 (2008), Nr. 6, S. 712–726

[SNO14] STENTZEL, Tom ; NIKNAM, Masoud ; OVTCHAROVA, Jivka: Comparison framework for PLM maturity models. In: *IFIP International Conference on Product Lifecycle Management* Springer, Berlin, Heidelberg, 2014, S. 355–364

[SO+12] STRAHILOV, Anton ; OVTCHAROVA, Jivka u. a.: Mechanische Absicherung automatisierter Montageanlagen mit physikbasierten Simulationen. In: *DFX 2012: Proceedings of the 23rd Symposium Design For X, Bamberg/Erlangen, Germany 04.-05.10. 2012*, 2012

[SOB12] STRAHILOV, Anton ; OVTCHAROVA, Jivka ; BÄR, Thomas: Development of the physics-based assembly system model for the mechatronic validation of automated assembly systems. In: *Proceedings of the Winter Simulation Conference* Winter Simulation Conference, Berlin, 2012, S. 63

[SOW09] STANEV, Stilian ; OVTCHAROVA, Jivka ; WALLA, Waldemar: Formal Method for Validation of Product Design through Knowledge Modelling. In: *International Conference on Knowledge Engineering and Ontology Development (KEOD)*, 2009, S. 166–170

[SR97] SIDDIQUE, Zahed ; ROSEN, David W.: A virtual prototyping approach to product disassembly reasoning. In: *Computer-Aided Design* 29 (1997), Nr. 12, S. 847–860

[SSKLK13] SHREINER, Dave ; SELLERS, Graham ; KESSENICH, John ; LICEA-KANE, Bill: *OpenGL programming guide: The Official guide to learning OpenGL, version 4.3*. Addison-Wesley Professional, Boston, USA, 2013

[Sto04] STOYANOV, Stoyan: *Street Fighter Racing Concept*, Angel Kanchev University of Ruse, Bachelorthesis, 2004

[Str06] STROUD, Ian: *Boundary representation modelling techniques*. Springer Science & Business Media, Berlin, 2006

[SWA+10] STANEV, Stilian ; WALLA, Waldemar ; AWAD, Ramez ; BITTEL, Vitalis ; OVTCHAROVA, Jivka: Lifecycle oriented information model to support the production driven product validation. In: *Proceedings of the 6th CIRP-Sponsored International Conference on Digital Enterprise Technology* Springer, Berlin, Heidelberg, 2010, S. 277–288

[SWO19] SCHNEIDER, Georg F. ; WICAKSONO, Hendro ; OVTCHAROVA, Jivka: Virtual engineering of cyber-physical automation systems: The case of control logic. In: *Advanced Engineering Informatics* 39 (2019), S. 127–143

[Tec] TECHVIZ: *TechViz XL Solution*. https://www.techviz.net/de/techviz-xl, . – Accessed: 2019-04-14

[Uni] UNITY3D: *Unity3D Engine*. https://unity.com, . – Accessed: 2019-04-14

[Unr] UNREAL: *Unreal Engine*. https://www.unrealengine.com, . – Accessed: 2019-04-14

[Vll19] VLLASA, Qendrim: *Automatisierte Virtualisierung von Anlagen durch Zusammenführung von ECAD, MCAD und Programmierung*, Karlsruhe Institute of Technology, Institute for Information Management in Engineering, Masterthesis, 2019

[WBH+16] WEISER, Ann-Katrin ; BAASNER, Bernd ; HOSCH, Manfred ; SCHLUETER, Meike ; OVTCHAROVA, Jivka: Complexity assessment of modular product families. In: *Procedia CIRP* 50 (2016), S. 595–600

[WBO11] WALLA, Waldemar P. ; BAER, Thomas ; OVTCHAROVA, Jivka: Impact of Modularised Production on Product Design in Automotive Industry. In: *DS 68-5: Proceedings of the 18th International Conference on Engineering Design (ICED 11), Impacting Society through Engineering Design, Vol. 5: Design for X/Design to X, Lyngby/Copenhagen, Denmark, 15.-19.08. 2011*, 2011

[WBO13] WICAKSONO, Hendro ; BELZNER, Tim ; OVTCHAROVA, Jivka: Efficient energy performance indicators for different level of production organizations in manufacturing companies. In: *IFIP International Conference on Advances in Production Management Systems* Springer, Berlin, Heidelberg, 2013, S. 249–256

[WBO15] WEISER, A ; BAASNER, B ; OVTCHAROVA, Jivka: Necessity of the consideration of strategic aspects in variant decisions of modular product architectures. In: *2015 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)* IEEE, 2015, S. 1551–1555

[WDKHR13] WICAKSONO, Hendro ; DOBREVA (KRAHTOVA), Preslava ; HÄFNER, Polina ; ROGALSKI, Sven: Ontology Development towards Expressive and Reasoning-enabled Building Information Model for an Intelligent Energy Management System. In: *5th International Conference Knowledge Engineering and Ontology Development, 2013, Vilamoura, Algarve, Portugal*, 2013

[WJRO14] WICAKSONO, Hendro ; JOST, Fabian ; ROGALSKI, Sven ; OVTCHAROVA, Jivka: Energy efficiency evaluation in manufacturing through an ontology-represented knowledge base. In: *Intelligent Systems in Accounting, Finance and Management* 21 (2014), Nr. 1, S. 59–69

[WO12] WICAKSONO, Hendro ; OVTCHAROVA, Jivka: Energy Consumption Regulation and Optimization in Discrete Manufacturing through Semi-automatic Knowledge Generation using Data Mining. In: *10th Global Conference of Sustainable Manufacturing (GCSM), Istanbul, Turkey*, 2012

[WRO12a] WICAKSONO, Hendro ; ROGALSKI, Sven ; OVTCHAROVA, Jivka: Knowledge Management Approach to improve Energy Efficiency in Small Medium Enterprises. In: *10th International Conference on Manufacturing Research (ICMR 2012), Birmingham, UK*, 2012

[WRO12b] WICAKSONO, Hendro ; ROGALSKI, Sven ; OVTCHAROVA, Jivka: Ontology Driven Approach for Intelligent Energy Management in Discrete Manufacturing. In: *4th International Conference on Knowledge Engineering and Ontology Development (KEOD)*, 2012, S. 108–114

[WSR+11] WICAKSONO, Hendro ; SCHUBERT, Viktor ; ROGALSKI, Sven ; LAYDI, Youssef A. ; OVTCHAROVA, Jivka: Ontology-driven Requirements Elicitation in Product Configuration Systems. In: *Enabling Manufacturing Competitiveness and Economic Sustainability, ElMaraghy, H. A. (Ed.), Springer Verlag, Berlin, Heidelberg*, 2011, S. 63–67

[ZBO12] ZEHNTER, Christina ; BURGER, Alexander ; OVTCHAROVA, Jivka: *Key-Performance-Analyse von Methoden des Anforderungsmanagements*. Bd. 7620. KIT Scientific Publishing, Karlsruhe, 2012