IMET - International Conference on Interactive Media, Smart Systems and Emerging Technologies (2023)
N. Pelechano, F. Liarokapis, D. Rohmer, and A. Asadipour (Editors)

Programmatic Design and Architecture of an Immersive Laser Laboratory

A. Müller[1,2], S. Müller[2], T. Brixner[2], and S. v. Mammen[1]

[1]Julius-Maximilians University, Chair of Human-Computer Interaction, Games Engineering, Germany
[2]Julius-Maximilians University, Institute of Physical and Theoretical Chemistry, Germany

Abstract

femtoPro provides an immersive laser laboratory experience for training, experimentation and analysis. In this paper, we present its programmatic design and architecture driven by requirements such as: accurate (non-)linear optics calculations, real-time performance on virtual reality (VR) headsets, easy authoring capabilities of pedagogical "quests", as well as natural user interactions and effective, multisensory feedback. We elaborate on how we tackle these challenges considering frontend components and the simulation backend. Notable features encompass an incremental, graph-based laser path solver algorithm with caching capabilities, a volumetric pulse shape renderer with accurately mapped color and transparency, an intuitive interaction system that translates coarse hand motions into precise adjustments, a comprehensive interface for configuring, conducting, and analyzing complex experiments with double precision accuracy including time-series measurements with moving platforms, and a generally flexible, modular architecture achieved through event-based communication, dynamic binding, and other state-of-the-art coding principles. Most of our proposed solutions should be directly applicable to similar immersive science laboratories, independently of their concrete domain.

CCS Concepts

• ***Computing methodologies*** → *Modeling and simulation;* • ***Applied computing*** → *Physical sciences and engineering;*

1. Introduction

Laser technologies are becoming increasingly ubiquitous in our everyday lives, with numerous applications across various fields, such as research, industry, commerce, medicine, and defense [SPI]. Moreover, lasers with ultrashort pulse durations in the order of femtoseconds (10^{-15}s) and below are used for time-resolved spectroscopy of quantum dynamics in physics, chemistry, biology, and material sciences. However, setting up and maintaining an ultrafast laser laboratory requires significant resources, such as sophisticated equipment, technical and physical infrastructure as well as experienced personnel. The high initial costs, ongoing resource needs and limited capacities can be a barrier for many institutions, restricting their ability to advance on this rapidly evolving technology. The femtoPro software [BvMMM] aims at accessibility, effectiveness, and efficiency of teaching, conducting and analyzing ultrafast laser experiments by providing an interactive, eye-safe, collaborative VR laboratory experience backed by a realistic optical simulation model. It tackles these challenges by offering an immersive experience featuring a lab space and 3DUI elements that align very closely with the real-world environment and instruments, various feedback mechanics tailored towards quick apprehension of laboratory procedures and the physical system including a sophisticated quest system and an interactive whiteboard. It is optimised to yield high, constant frame rates to run on standalone VR headsets to maximise flexibility and user experience. femtoPro has been successfully integrated into an elective course on "Ultrafast Spectroscopy and Quantum Control" for the chemistry Master's program at the University of Wuerzburg. Its apt use for outreach initiatives and engaging a wider audience has successfully been demonstrated at public science fairs and academic conferences. Beyond its current use cases, femtoPro can serve as a versatile tool for designing, simulating, and conveying standard and custom optical experiment setups in various fields of application [BMM*23]. This paper provides an overview of femtoPro's software architecture, encompassing the system's components and their interactions, the management and processing of general and simulation data as well as the underlying rationale behind UI and software design decisions. By presenting this information, the paper offers insights and knowledge that can be transferred to the development of virtual laboratories and other types of experiment-based interactive simulation experiences in different fields of science. For readers interested in the details of the implemented optics model and its mathematical foundations, we published another paper which focuses on femtoPro's pulse transformation algorithm approximating linear and non-linear optical responses as well as the various integrated aspects of light-matter interactions [BMM*23]. Moving forward to Section 2, we will examine related work on VR laboratory experiences in different domains and more specifically for optics

DOI: 10.2312/imet.20231256

applications. Section 3 outlines a typical application scenario for femtoPro. This provides the basis for Section 4 which delves into the software's architecture, focusing on its core concepts, namely the experiment instance and the simulation system. Section 5 concludes the paper and outlines planned extensions of femtoPro.

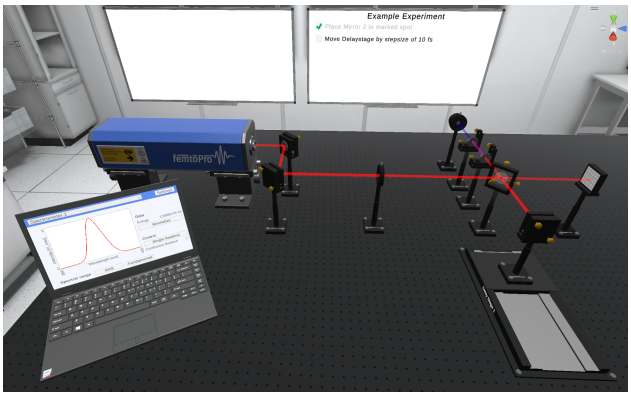

Figure 1: *General overview. In-game screenshot.*

2. Related Work

In computer graphics, ray tracing is a well-known rendering technique used to create realistic images by simulating the way light interacts with objects in a scene. It can also be applied in optics system design to predict and optimize the path of light beams [SM62, Wik]. Furthermore, various solutions exist for numerically solving Maxwell's equations, which are a set of partial differential equations that can accurately describe the behavior of light waves, and available solvers often take into account the interaction with quantum mechanical systems [ORI*10, Lum, MPBK09, Man, KMP]. In this context, the UI of the LabView platform developed by Schmidt and colleagues was an inspiration for the femtoPro project [SHSF]. However, solutions employing the aforementioned methods do not meet the requirements for a VR laser lab, as ray tracing does not consider the wave nature of light essential to specific laser experiments or the lateral spatial beam profile, and solving Maxwell's equations are too costly for real-time computation. Numerous applications already exist in the realm of VR training, e.g., in the health sciences including dentistry [VMWOD15], first-aid [BBRvM19], and classroom management [LLH*16]. VR training applications also span across diverse engineering fields, including construction [WWW*18], surveying [BC19], and electrical engineering [JG18]. To our knowledge, the first VR laser lab in the field of optics was reported in 1996, introducing students to the principles of light wave propagation [BMK96]. Numerous solutions have been following in relevant areas, e.g. fiber optics engineering [HTRK13], and experimental teaching [TGYP16, QLF*18]. Additionally, some commercial products are under development or have already been released, such as the Immersive Photonics Lab, a VR application stating to "accurately replicate physical phenomena and allow for practical skill development in industrial or educational contexts" [VR , ALP]. femtoPro's unique value proposition lies in its ability to create a realistic and immersive virtual laboratory experience, combining essential features such as the alignment of different optical devices, accurate simulation and analysis of various non-linear optical phenomena, an easy-to-use didactic content editing system, and remote collaboration (in development for a future version). femtoPro takes advantage of the widely used Unity game engine's pre-built capabilities in rendering, detection of collisions, specification of interactions, sensory processing, and networking [HSH*20].

3. Overview

Fig. 1 depicts a screenshot of the femtoPro laser laboratory. As in real-world labs, the experiment table is its central element. A red beam is emitted from the blue laser device to the left of the image. In the given situation, the beam propagates through nine optical elements that are mounted to the table and serve different purposes such as reflection, refraction, frequency conversion, wavelength filtering, and detection. In the bottom-right corner of the image, an optical element is mounted to a mechanical delay stage that allows users to incrementally change the path length of the laser beam to generate time series of differently parameterized measurements. The way the laser propagates is simulated based on Gaussian beam optics, taking into account additional femtosecond laser pulse properties such as dispersion [BMM*23]. The simulation's outcome not only determines the trajectories of the beams but also their optical properties measured in the virtual environment and plotted on the screen of the laptop at the bottom-left of the screenshot. The user interface of the measurement application running on the virtual laptop provides the same functionality as the software applications used in real-world laboratories. On the wall at the back of the virtual lab, there are two whiteboards which are primarily used to inform the user about open and completed quests in training curricula. femtoPro is developed for stand-alone head-mounted displays such as the Quest and the Pico device series [Met, PIC]. They have all the required sensing, processing and rendering capabilities built-in, provide the user with a stereoscopic view with six degrees of freedom, and feature a pair of 3D controllers. femtoPro can both be used seated or in standing pose. There are various quests, which introduce the user to navigation in the VR environment by means of teleportation, selection, placement and configuration of optical elements, the use of support devices such as the delay stage, the virtual laptop, and, very importantly, the adherence to safety guidelines during all experiment stages. There is an extensive curriculum of additional quests that teach the user to not only setup and run more involved experiments but also to incrementally adjust and expand those.

4. Architecture

In this section, we provide an overview of the femtoPro's software architecture as illustrated in Fig. 2. The concrete implementation adheres to industry standards and best practices, such as clean code principles and object orientation, which may not be apparent in the architectural representation [Mar17]. Furthermore, by using agile development methodologies, we prioritize adaptability, and responsiveness to changing requirements over rigid planning and upfront design [Mar17]. In the figure, grey boxes represent components within the *Experiment Instance* or *Simulation System*. Black arrows indicate the simulation data flow, and blue dotted lines the

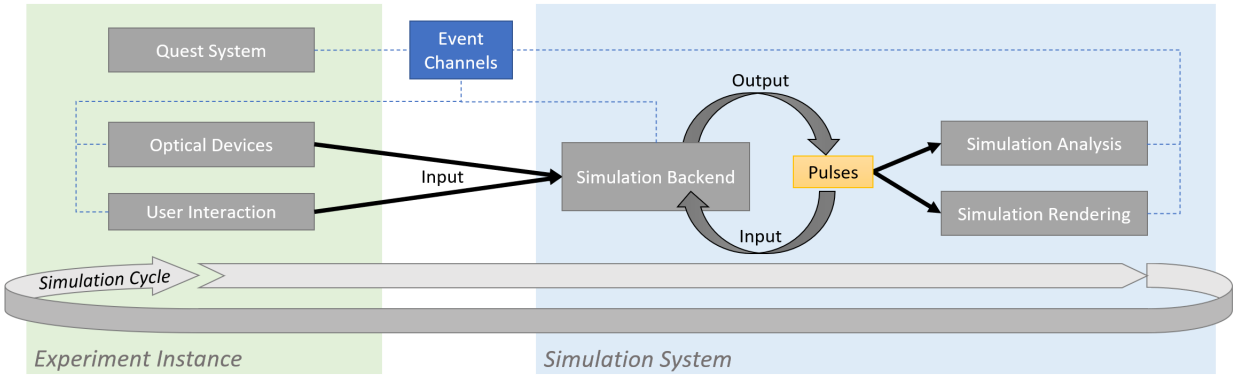

Figure 2: *Schematic overview of the system's architecture.*

overall communication architecture. The experiment instance section comprises *Optical Devices* along with *User Interactions* to manipulate them. Additionally, it includes a *Quest System* to define experiment-specific quests. The simulation system can process numerical input data from any experiment instance. It comprises three components: *Simulation Backend*, *Analysis*, and *Rendering*. At the beginning of each simulation cycle, the backend receives input data from the optical devices of an experiment instance, including properties like their position. It then calculates the properties of all laser pulses. As indicated in the *Input-Output* cycle, and described in more depth in Section 4.2, the calculations for the laser path are performed incrementally for each optical device, where one step's result serves as input for the next, until the entire path is determined in a self-consistent manner. The generated pulses are passed on to *Simulation Rendering* and *Analysis* components. The first generates a visual representation of each laser pulse based on its numerical characteristics. The latter is responsible for providing real-time graphical representations of aggregated simulation data such as curves or bar charts. The communication within and between the components of the system is realized by means of independent event channels that follow the publish-subscribe pattern [GHJV95]. In femtoPro, event channels are implemented as wrapper classes for standard C# events, which manage all event handling logic. Any component that holds a reference to a channel can invoke or listen to the wrapped event. These references to channels are injected into the respective publishers and subscribers in the editor via drag and drop or programmatically via script – a technique known as dependency injection [Mar17]. This principle plays a crucial role in ensuring that the project remains decoupled and maintainable. Event binding can also happen dynamically during runtime, thereby providing greater flexibility in the design of interactive UI elements. For example, when a user clicks on a measurement device in the UI for analysis during runtime, events of the desired device are injected into the respective UI panel. Our architectural decision to separate the simulation system and experiment instance led us to use a multiscene workflow for the concrete implementation. In this approach, the simulation logic and elements that are generic and not specific to any particular experiment, such as the lab environment, are located in the main scene. A specific experiment setup, on the other hand, is stored in a separate scene, which can be loaded additively to the main scene. We opted for this approach over a single scene workflow for several reasons. Firstly, it offers better scalability as we plan the creation of 50+ new experiments in the near future. Secondly, it lowers initial loading times and memory consumption at runtime. Finally, collaboration using version control is improved by reducing merge conflicts since experiment and simulation data are located in separate scene files. The only disadvantage of the multiscene workflow is that references between scenes become invalid once any affected scene unloads. We address the issue by restricting the placement of elements to the scene where they are (mostly) referenced, thus minimizing cross-scene references. If required, we pass references across scene boundaries through events at runtime.

4.1. Experiment Instance

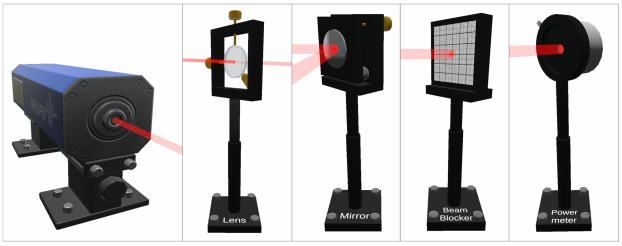

Figure 3: *Examples of optical devices. From left to right: laser source, lens, mirror, beam blocker and power meter. In-game screenshots, retouched for visual clarity.*

Optical devices provide functionality such as light manipulation, detection, and measurement. For femtoPro we have implemented a wide range of optical elements that can be categorized as laser sources, pulse transformers, and pulse end points. Pulse transformers are optical elements that generate new outgoing pulses at their surface through linear and non-linear interactions with incoming pulses. These interactions can involve various optical phenomena such as reflection or refraction which enable the pulse transformer to change the spatial as well as spectral-temporal properties of the incoming pulse. Examples of pulse transformers include mirrors or lenses (see Figure 3). In contrast to pulse transformers, pulse endpoints are physical elements that mark the end of a laser pulse's

path. Pulse endpoints can be sensors such as power meters (see Fig. 3) or spectrometers that measure the wavelength-dependent power distribution of a pulse at their surface. The laboratory environment and interactive components such as the laptop as well as the player's hands also serve as pulse endpoints. For the femtoPro simulation, linear and non-linear interactions of laser pulses are only tracked at pulse transformer surfaces. All other surfaces terminate the laser beam. All optical devices have distinct geometric and optical properties that can be modified in the game engine's editor as well as in real time during gameplay impacting the simulation output. Examples are the adjustment of the curvature radius of a mirror or the definition of the refractive index of a lens determining a refracted pulse's direction and beam divergence properties. Furthermore, the settings for each laser source can be customized to allow the generation of pulses with varying characteristics including different wavelength, intensity and duration. The ability to configure and customize optical components is crucial to produce experiments that cover a wide range of optical phenomena. We have included detailed properties that can be modified in the editor and at runtime. For VR contexts, it is important that we maintain immersion, avoid discomfort such as motion sickness, and that the usage of optical devices is intuitive and self-explanatory. To achieve these goals, we used industry standards for player locomotion and interaction with the game environment. In addition, in order to closely mimic the interaction procedures of real lab work, we also developed custom interaction techniques for the optical devices.

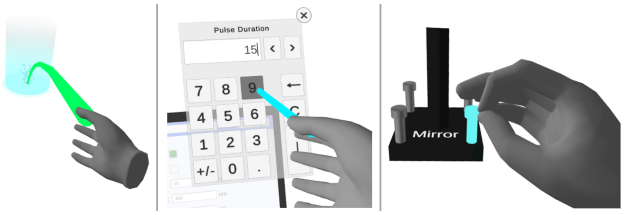

Figure 4: *Interactions from left to right: teleporting, interacting with the keypad UI and mounting an optical device onto a surface. In-game screenshots, retouched for visual clarity.*

To move to a new location, the player can use the teleportation system by selecting a destination with the thumbstick of the controller and releasing it to teleport. Valid, unobstructed destination points are indicated by a trajectory curve from the current position to the destination that is also marked with a circle on the floor, providing clear feedback to the user (Fig. 4, left). To improve comfort in a seated VR experience with limited space available, the player can also turn left or right by 45 degrees using the corresponding thumbstick action. We completely avoided Heads-Up Displays (HUDs), i.e., UI elements that are overlaid on a fixed position on the player's screen. Instead, user interfaces are projected onto elements in the virtual world, such as a virtual laptop or a level selection folder. This design approach allowed for a more seamless integration of the user interface into the virtual world, thus promoting a more immersive experience while reducing visual discomfort. We utilized the engine's standard raycast system to select UI elements, to click buttons, and drag sliders. As a visual cue of their interactivity, specific UI elements are highlighted upon hover. We also implemented a virtual keyboard and numpad for inputting strings and numbers into fields (Fig. 4, middle). They open in front of the clicked input field similar to a real-world keyboard. The user simply clicks on corresponding keys to enter characters, or to use additional comfort features like erasing all input. While inputting numbers or strings via virtual keyboard is an industry standard, it can still be cumbersome for users. To improve usability, it is worth considering the addition of a speech-to-text feature in the future. To increase the realism, we used animated hand meshes instead of the default controller model and deployed grabbing and pinching as two default interactions. When a user's hand approaches an interactive object, its mesh is highlighted with a glow, intuitively guiding the user towards interaction options (Fig. 4, right). The user can grab optical devices and other interactive objects, such as the virtual laptop and level selection folder. By default, grabbing involves a force pull mechanism that pulls elements towards the controller from a distance. However, this reduces realism and makes precise positioning difficult. To address this, we disabled force pull to restrict grabbing interactions to objects within close proximity to the user's hand. Every optical device can be locked in place on the table to prevent accidental grabbing interaction. The position of the screws attached to the floorplate of each device indicates whether it is locked or not. The device can be grabbed and lifted when the screws are up, and it is locked when the screws are down. The user can change the locking state of the device by pinching its floor screws (Fig. 4, right). Besides that, the pinching gesture is used to precisely adjust optical devices, for example by interacting with their micrometer adjustment screws. To address the insufficient precision of user input, we developed a crank mechanism that translates coarse circular arm motions to the desired precise adjustments of screws and device components. This mimics a real-world device frame whose position can likewise be fine-tuned by respective screws.

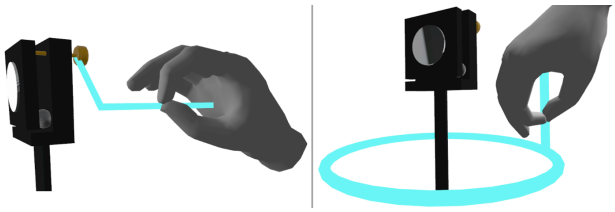

Figure 5: *Fine-adjusting the angle of a mirror (left) and its upper stand (right). In-game screenshots, retouched for visual clarity.*

As shown in Fig. 5 (left), the horizontal and vertical angle of a mirror's frame can be adjusted in this way. Here, a complete revolution of the relevant screw causes it to move by 0.25mm which results in a 0.286-degree rotation of the frame. Figure 5 (right) illustrates the fine adjustment of the upper stand. Dependent on the type of interaction, an L-shaped line or a circle is rendered relative to the user's hand and the interactable's center, visually guiding the user's motion in an intuitive manner. The precision of input increases with the hand's distance from the object being manipulated. This is because a greater distance means a larger circular motion of the hand, translating into a smaller and therefore more precise angular change in the object's position. Furthermore, users receive direct feedback on the validity of their interactions. For example, a continuous, subtle haptic impulse is given for each step the screw is turned, whereas an intense single vibration impulse is triggered and the circle changes to red when the device reaches a physical threshold. The quest system plays a vital role in delivering educational

content. It aims at teaching the fundamentals of the VR application and guides the users through the process of conducting and analyzing experiments according to expert-approved procedures and safety guidelines. An according quest manager object is attached to every experiment scene, that can hold multiple quests which, in turn, can contain multiple conditions for flexible quest design. A quest where the user moves an object through several checkpoints can, for instance, be easily modelled by means of multiple conditions. A quest is fulfilled when all its conditions are met. Similarly, the scene's overall mission is fulfilled when all quests within that scene are accomplished. The fulfillment state is transmitted via events to respective listeners that are bound to the instantiation of the experiment scene at runtime. This dynamic binding results in modular and interchangeable components, thereby enhancing the flexibility of the quest system. An optional *in sequence* parameter ensures that certain conditions can only be met in a specific order, as, for instance, dictated by safety guidelines and standard procedures.

Figure 6: *Quest editor (left) and the automatically generated Quest UI displayed in-game on a virtual whiteboard (right). Unity Editor screenshots. Retouched for better readability.*

The quest system has been specifically designed for physics experts who have little programming experience. A quest's metadata such as its name or the in sequence-attribute can be defined without any coding by using the UI of the editor (Figure 6, left). The designer can choose from a wide range of pre-made condition scripts suitable for most situations. In these cases, the user only needs to setup the desired references. For instance, when using the generic condition for the laser hitting an object, the reference to the specific object must be assigned to the corresponding field inside the editor. Furthermore, the designer can extend existing or create entirely new condition scripts easily without the need to know the quest system's internal logic, as it is encapsulated in abstract base classes. Inside the virtual laboratory, a whiteboard displays the titles and descriptions of all available quests in the current experiment scene. A checkbox is attached to each quest text to clearly indicate its current fulfillment state (Figure 6, right). Relevant events of the quest system, like their initialization and change of state, are transmitted to the whiteboard UI via event channels (as described in Section 4). When a quest is fulfilled, its description fades to grey, the checkbox gets a green checkmark and an uplifting bell-sound is played. This immediate audio-visual feedback has proven to be an effective reward mechanic that contributes to an enjoyable user experience [RSBLW17].

4.2. Simulation System

The simulation system's main objective is to generate laser pulses using numerical input provided by optical devices and user interaction (simulation backend), which can then be rendered in 3D (simulation rendering) and analyzed at runtime during gameplay (simulation analysis). The simulation backend is responsible for generating the complete path of all laser pulses, which originates from a laser source, traverses various optical devices, such as lenses and mirrors, until it reaches its termination point, such as a measurement device or a wall. We have modeled the path of laser beams as a directed graph to facilitate calculations. This model features laser sources as root nodes, optical devices with the ability to transform pulses as internal nodes, and pulse endpoints as leaf nodes. Laser pulses serve as directed edges connecting the nodes, with each pulse having exactly one source and target node. As depicted in Algorithm 1, the calculation of laser paths is performed each frame in a node-by-node manner by processing a queue containing all nodes requiring (non-)linear optical transformations of incoming to outgoing pulses.

Algorithm 1 Pseudocode of the laser path solver algorithm.

```
1: At experiment initialization:
2: Add all source nodes to process queue
3: For each frame:
4: Add all nodes affected by change or pulse interruption
5: while queue !empty && increment++ < MAX
6:   node = queue.Dequeue()
7:   pulses_out = Transform(node, node.pulses_in)
8:   Trace target nodes of all pulses_out
9:   Add target nodes of new, changed & vanished pulses
```

The first step of the algorithm is to add all laser sources to the queue once an experiment scene is loaded. Each frame, the algorithm iterates through all optical elements, or nodes, of the experiment. If a node's optical or geometrical properties have changed in comparison to the last frame, the node itself and all nodes that are directly affected by this change, i.e., all source nodes of incoming edges, are added to the queue. An algorithm to detect any pulse interruptions is executed next. In case of interruptions, both source and target node of the affected pulse are added to the queue. It is worth mentioning that the process queue can contain only unique node entries and rejects any duplicate insertions. After enqueuing all nodes affected by change or pulse interruption, the queue is processed: For each dequeued node the transformations of any incoming pulses are calculated based on the principles of (non-)linear optics. The resulting outgoing pulses are traced and their target nodes are determined. Target nodes of pulses that have changed since last frame are subsequently added to the queue. The laser path solver algorithm is terminated when either the queue is empty or when the maximum iteration count for a frame is reached. The latter condition acts as a safeguard against infinite looping due to potential cycles in the laser path, for example when a specific circular arrangement of mirrors reflects a beam indefinitely. Dependent on the complexity of an experiment setup, accurately calculating (non-)linear optical pulse transformations can be the most costly step of the laser path solver. By calculating only those parts of the graph that are affected by change, and by preserving the rest as is, we increased the performance of the whole simulation without compromising accuracy. Furthermore, the algorithm immediately yields all relevant changes in its environment, including geometrical and optical properties of nodes, as well as pulse-matter interactions, contributing to a highly authentic laboratory experience. While the laser path solv-

ing algorithm minimizes unnecessary computations, some experiment configurations may require revisiting nodes within the same frame. This issue is an inevitable consequence of the underlying graph traversal logic and can arise, for instance, when a laser beam is reflected back to its originating node. On average, only a part of the laser path changes, whereas computational cost tends to decrease the closer these modifications are to a leaf node. Besides that, the impact on performance is significantly greater for path segments that comprise non-linear optical pulse transformations compared to those limited to linear optics. In the worst-case scenario, the alteration of root node properties demands the recalculation of the entire graph. In practice, a performance analysis conducted during VR play sessions underscored the importance of path caching by revealing that in most frames the laser path remained largely unchanged. This can be attributed to the user engaging in various activities that do not interfere with the simulation, such as reading quest descriptions, exploring the environment, inspecting the experiment setup, or analyzing simulation results on the virtual laptop. For treating the spatial behavior of laser beams we implemented a model of Gaussian beam propagation [BMM*23]. To keep the cost low, we decided to ignore astigmatism such that all beams have circular cross sections. We also ignore diffraction effects when a beam partially hits an obstacle. Instead, we approximate the transmitted beam as a perfectly circular Gaussian beam whose diameter is defined by the common geometric cross-section of incident beam and impacted surface (Figure 7, right). The pulse

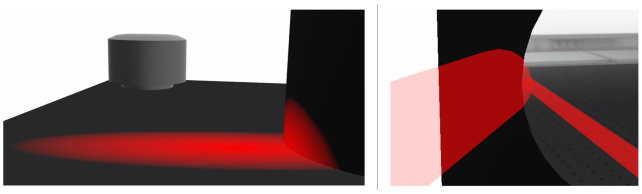

Figure 7: *Left: accurate projection of a beam's cross section onto a pedestal. Right: rendering of a partially reflected beam. In-game screenshot.*

interruption detection algorithm is executed once per frame for every pulse, resulting in a high iteration count. For performance reasons, instead of tracking collisions with the actual volumetric shape of the beam, a simple line is cast from the center of its origin to the surface of the hit target. If the length of the line is smaller than in the previous frame, the pulse is considered interrupted. On the downside, this approach can only detect intersections by objects that at least cross the central axis of the beam resulting in specific partial obstructions not being properly identified. However, partial interactions of beams with non-optical elements, such as the user's hand, have negligible relevance to the experiment analysis. A potential impact may be on the perceived realism of the VR experience, but as the beam radius is limited by design to a few centimeters due to safety requirements, inaccurate details of volumetric features are not visually noticeable on a VR device. Most available game engines currently utilize float precision for internal physics calculations to simulate the movement and interactions of objects within the game environment. With a maximum beam propagation length of 10m, achieving a precision of approximately one hundredth of a wavelength, denoted as $\frac{\lambda}{100}$, is necessary to accurately observe the interference pattern generated by overlapping pulses. The wavelength of femtosecond pulses is around 500nm which consequently requires a precision of 5nm ($5 \cdot 10^{-9}$m). Despite that the C# float datatype encompasses the desired range (10^{-45} to 10^{38}), it is still inadequate. This is because it can only guarantee accuracy without floating point errors for numbers with a maximum of 9 significant digits, which is one digit less than required [Mic]. Therefore, the pulse transformation requires positional data of optical devices and pulses to be in double precision to simulate interferometric optical phenomena with sufficient accuracy (Fig. 8). We addressed this

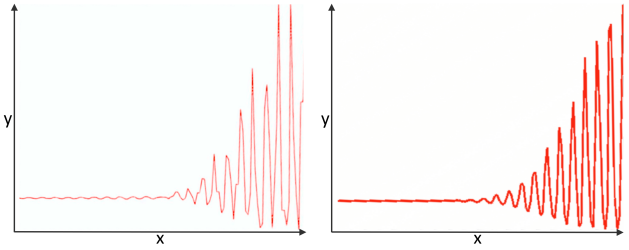

Figure 8: *Inaccurate (left) vs. accurate (right) depiction of pulse interference patterns with float vs. double precision positional data. The y axis denotes the intensity of overlapping pulses measured at specific sampling points (x axis). In-game screenshots, retouched for visual clarity.*

issue by storing and maintaining the positional data of optical devices in both float and double precision, using the latter specifically for the pulse transformation calculations. When an optical device is not fixed to the table or other surfaces, its location and orientation is provided by the physics engine as float values. These values are upcast to double precision upon change caused by external forces such as gravity, pushing, or pulling. When locked, an optical device is unaffected by physics. Its positional data can only be altered indirectly through alignment screws or the position of the object it is attached to, such as a motorized platform. The crank-like interaction system used for screw rotation maps coarse movements to fine adjustments, meeting the accuracy requirements of the simulation. Likewise, the linear movement steps of a motorized platform is defined in double precision through numeric input fields of the user interface, allowing to derive the accurate position of any optical elements attached to it. This approach ensures the required simulation accuracy while leveraging the performance and convenience of the float-based physics engine for coarse positioning and other interactions, effectively combining the best of both worlds. The simulation rendering system is responsible for visualizing the shapes of the pulses as well as their projection onto hit surfaces. Even though the actual propagation path of a laser beam through air is not visible to the naked eye, we chose to include its visualization as an optional feature to improve understanding of the simulation and enhance its educational value. The beam is represented as a simplified 3D mesh to balance performance and visual fidelity. Therefore, we approximated Gaussian-shaped beams by a mesh composed of linked rings with 16 vertex points placed at regular intervals. Additionally, a mesh clipping algorithm ensures that a pulse exactly ends at a hit surface, regardless of its incident angle and without any mesh overlaps or noticeable gaps between surface and pulse. The rendering of the beam's cross section is realized by

a performant shader program run on the GPU (Figure 7, left). Game engines usually provide specific projector materials, also called decals, for similar problems such as the projection of spotlights to a surface. However, we noticed a significant drop in performance when used for beams that constantly move, which is rather common during the alignment phase of an experiment. As another drawback, the rendering capabilities of standard projector materials are very limited for mobile and VR platform development. It is important to note that the spectral composition of the beam determining its color is accurately mapped and displayed within the given 32-bit color space. Furthermore, the beam's intensity profile is visualized by a gradual decrease in opacity towards the edges of the pulse's volumetric shape and its cross-section. Both effects are realized with the shader script calculating color and alpha values for each pixel with a predefined look-up table as well as given pulse properties and positional data as input. We further implemented object pooling [Gre18] for pulse meshes to improve performance, particularly when a user interacts with the experiment. Instead of creating and destroying meshes every time the positions of pulses change during the alignment process, we simply update the vertex positions of reusable mesh instances stored inside a pool, resulting in a significant reduction of memory allocations. To further improve performance, the mesh and cross-section of a pulse are only updated when its properties have changed, preventing numerous unnecessary calculations in an unaltered state. During play-tests on the targeted standalone VR platforms, simulation rendering accounted for only a small portion of the entire frame time – even during peak workload with continuous changes to the entire laser path – suggesting that it has been optimized effectively for those platforms.

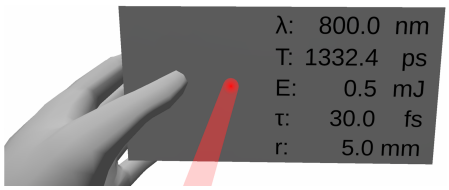

Figure 9: *Laser card showing the properties of an incident pulse. In-game screenshot, retouched for visual clarity.*

In accordance with standard lab procedures, a laser alignment card with non-reflective coating can be used to detect the presence and location of a laser beam. To foster learning, we augmented the laser card with optional analysis capabilities: the properties of a hit pulse at its intersection are displayed on the card itself as numerical values that are automatically updated, rounded, and scaled (Figure 9). But the virtual laptop serves as the primary interface for detailed experiment configuration and analysis. Each laser source, measurement device, and motorized moving platform has its own UI panel that can be selected from a dropdown list in the header menu. It is important to highlight that all information displayed is gathered and processed in real time. This creates a responsive feedback loop, enhancing the user experience and promoting the effective assessment of experiments. The laser source UI panel allows for adjusting parameters of generated pulses, including their duration, wavelength, and intensity. With multiple laser sources present, each source can have its own individual settings, expanding the scope of possible experiment designs. The spectrometer is a sensing device that measures the intensity of incident pulses across a range of wavelengths. The spectrometer UI panel presents this spectral distribution in the form of a line graph (Fig. 10). Therein, the x axis represents a predefined range of wavelengths, and the y axis corresponds to the measured intensity at each respective wavelength. This information plays a crucial role in understanding the behavior of incident pulses, enabling users to properly align and evaluate specific experiments.

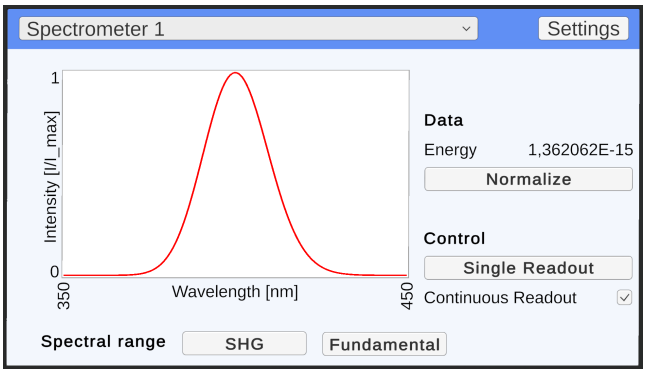

Figure 10: *UI panel: Spectrometer. In-game screenshot.*

The delay stage is a motorized platform that can be controlled to precisely move optical elements, such as mirrors or beam splitters, along a linear path. When the "Start" button is clicked on its UI panel (bottom-left of Fig. 11), the delay stage moves automatically in equidistant steps from a starting position to a destination according to the settings defined by the user, here converted to time delays using the speed of light. At each step, the delay stage pauses for the duration of a simulation cycle, enabling measurement devices such as a power meter to collect data for further analysis. Users can make coarse adjustments using sliders corresponding to the actual and projected positions on the delay stage (top-left in Fig. 11). Fine-tuning can be achieved using numerical input fields, which can display values in different orders of magnitude. However, internally all values are processed in the smallest available unit with double precision accuracy to ensure reliable simulation results (see Section 4.2). The interface also displays the output of measurement devices as a line graph for further analysis (bottom-right of Fig. 11). In this graph, the x axis represents the step number, while the y axis displays the measured value at each step, which in this case is pulse intensity. Overall, the interface enables users to comfortably conduct and analyze a series of measurements captured automatically at predefined sampling points, making it a valuable tool for accurately assessing experimental data.

5. Conclusion and Future work

In this paper, we detailed how the event system and the separation of simulation logic and experiment instances contribute to femtoPro's modular and maintainable project structure. We laborated on the importance of custom-tailored interaction mechanisms and UI panels to enhance productivity and authenticity. We explained how we struck a balance between performance demands and simulation

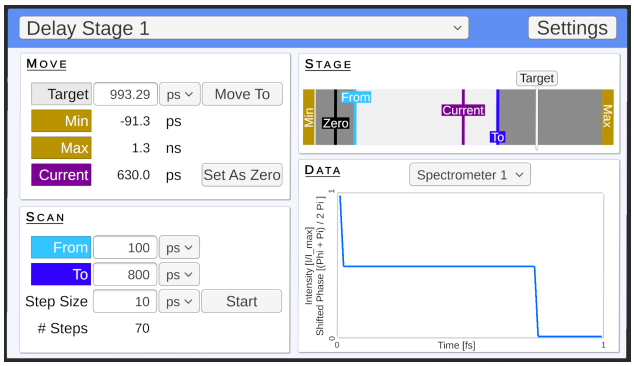

Figure 11: *UI panel: Delay Stage. In-game screenshot.*

accuracy in a real-time VR environment. Our approach involves simplifying aspects not critical to didactic objectives, such as restricting optical calculations to the experiment setup only, applying conservative optimization methods like path caching for resource-intensive core features, and enhancing accuracy where necessary, e.g. relying on double precision numerals for the fine-alignment of optical devices. Looking ahead, we aim to enhance femtoPro's educational value by incorporating a multiplayer feature that enables cloud collaboration and remote assistance. We also have already gathered extensive usage data that we want to evaluate with respect to educational benefits and shortcomings. In order to better harness the simulation model and to foster free exploration, we also plan to introduce a save game system and a fabricator for in-game creation and customization of optical elements. We are also working on the integration of more resource-intensive non-linear optics that incorporates resonant four-wave mixing processes such that time-resolved spectroscopy with molecular systems could be modeled.

References

[ALP] ALPHANOV: Virtual reality: Immersive photonics lab. https://www.alphanov.com/en/products-services/virtual-reality. 2

[BBRvM19] BUCHER K., BLOME T., RUDOLPH S., VON MAMMEN S.: VReanimate II: training first aid and reanimation in virtual reality. *Journal of Computers in Education 6* (2019), 53–78. 2

[BC19] BOLKAS D., CHIAMPI J. D.: Enhancing experience and learning of first-year surveying engineering students with immersive virtual reality. In *2019 FYEE Conference* (2019). 2

[BMK96] BROWN D. J., MIKROPOULOS T. A., KERR S. J.: A virtual laser physics laboratory. *VR in the Schools 2*, 3 (1996), 3–7. 2

[BMM*23] BRIXNER T., MUELLER S., MÜLLER A., KNOTE A., SCHNEPP W., TRUMAN S., VETTER A., VON MAMMEN S.: femtoPro: virtual-reality interactive training simulator of an ultrafast laser laboratory. *Applied Physics B 129*, 5 (2023), 78. 1, 2, 6

[BvMMM] BRIXNER T., VON MAMMEN S., MUELLER S., MÜLLER A.: femtoPro. https://www.femtoPro.com. 1

[GHJV95] GAMMA E., HELM R., JOHNSON R., VLISSIDES J.: *Design patterns: elements of reusable object-oriented software*. Pearson Deutschland GmbH, 1995. 3

[Gre18] GREGORY J.: *Game engine architecture*. crc Press, 2018. 7

[HSH*20] HUSSAIN A., SHAKEEL H., HUSSAIN F., UDDIN N., GHOURI T. L.: Unity game development engine: A technical survey. *Univ. Sindh J. Inf. Commun. Technol 4* (2020), 73–81. 2

[HTRK13] HAYES D., TURCZYNSKI C., RICE J., KOZHEVNIKOV M.: Virtual-reality-based educational laboratories in fiber optic engineering. In *ETOP Proceedings* (2013), Optica Publishing Group, p. EWP40. 2

[JG18] JONES A., GOLUB M.: Effectiveness of current-generation virtual reality-based laboratories. In *ASEE annu. conf. expo.* (2018). 2

[KMP] KENNEWEG T., MÜLLER S., PFEIFFER W.: QDT – a matlab quantum dynamics toolbox. https://qd-toolbox.org. 2

[LLH*16] LUGRIN J.-L., LATOSCHIK M. E., HABEL M., ROTH D., SEUFERT C., GRAFE S.: Breaking bad behaviors: A new tool for learning classroom management using virtual reality. *Frontiers in ICT 3* (2016), 26. 2

[Lum] LUMERICAL: High-performance photonic simulation software. https://www.lumerical.com. 2

[Man] MANČAL T.: Quantarhei: Open quantum system theory for molecular systems. https://github.com/tmancal74/quantarhei. 2

[Mar17] MARTIN R. C.: *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall Press, USA, 2017. 2, 3

[Met] META PLATFORMS, INC.: Meta quest 2. https://www.meta.com/de/quest/products/quest-2. 2

[Mic] MICROSOFT CORPORATION: Floating-point numeric types (c# reference). https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/floating-point-numeric-types. 6

[MPBK09] MITRIĆ R., PETERSEN J., BONAČIĆ-KOUTECKÝ V.: Laser-field-induced surface-hopping method for the simulation and control of ultrafast photodynamics. *Physical Review A 79*, 5 (2009), 053416. 2

[ORI*10] OSKOOI A. F., ROUNDY D., IBANESCU M., BERMEL P., JOANNOPOULOS J. D., JOHNSON S. G.: MEEP: A flexible free-software package for electromagnetic simulations by the fdtd method. *Comput. Phys. Commun. 181*, 3 (2010), 687–702. 2

[PIC] PICO: Neo3 pro / pro eye. https://www.picoxr.com/de/products/neo3-pro-eye. 2

[QLF*18] QIAN G., LU Z., FUYANG Z., SHANDE L., NONGLIANG S.: Research on ultra-fast laser experiment simulation system based on virtual reality. *Exp. Sci. Technol. 16*, 6 (2018), 124–128. 2

[RSBLW17] RAVYSE W. S., SEUGNET BLIGNAUT A., LEENDERTZ V., WOOLNER A.: Success factors for serious games to enhance learning: a systematic review. *Virtual Reality 21* (2017), 31–58. 5

[SHSF] SCHMIDT B., HACKER M., STOBRAWA G., FEURER T.: LAB2 - the virtual femtosecond laboratory. http://www.lab2.de. 2

[SM62] SPENCER G., MURTY M.: General ray-tracing procedure. *JOSA 52*, 6 (1962), 672–678. 2

[SPI] SPIE: Optics and photonics industry report (2020). https://spie.org/news/2020-optics-and-photonics-industry-report. 1

[TGYP16] TITOV I., GLOTOV A., YELIZAROV A., PETROV V.: Standartization use case of solid-state laser lab and rf&microwave amplifier remote and virtual laboratories at labicom. *International Journal of Online Engineering 12*, 9 (2016), 47–51. 2

[VMWOD15] VON MAMMEN S., WEBER M., OPEL H.-H., DAVISON T.: Interactive multi-physics simulation for endodontic treatment. In *SpringSim (MSM)* (2015), pp. 36–40. 2

[VR] VR LAB: Virtual laser lab. https://www.vr-lab.nl/virtual-laser-lab. 2

[Wik] WIKIPEDIA: List of ray tracing software. https://w.wiki/7MHB. 2

[WWW*18] WANG P., WU P., WANG J., CHI H.-L., WANG X.: A critical review of the use of virtual reality in construction engineering education and training. *Int. J. Environ. Res. Public Health 15*, 6 (2018), 1204. 2