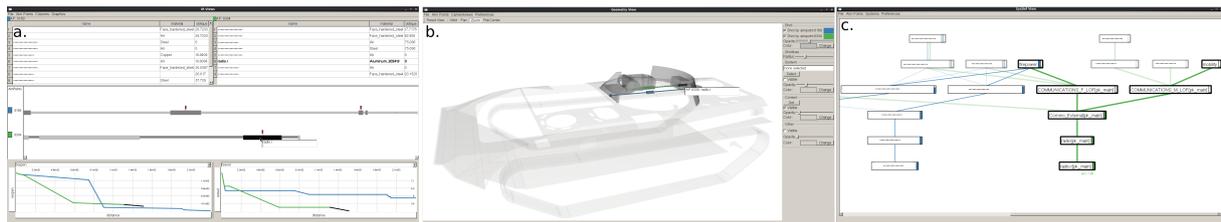


# A Shot at Visual Vulnerability Analysis

Ethan Kerzner<sup>1†</sup> Lee A. Butler<sup>2</sup> Charles Hansen<sup>1</sup> Miriah Meyer<sup>1</sup>

<sup>1</sup>University of Utah <sup>2</sup>US Army Research Laboratory



**Figure 1:** We present Shotviewer; software that supports visual analysis of spatial and nonspatial ballistic simulation data for vulnerability analysis. It consists of three linked views: a) the Shotline View displays an abstract representation of shots' paths through a vehicle; b) the Geometry View shows shots' 3D spatial context; and c) the System View visualizes the propagation of damage through a vehicle's systems. In this example, the green shot damages the vehicle's radio which impacts the vehicle's mobility and firepower capabilities.

## Abstract

Increasing the safety of vehicles is an important goal for vehicle manufacturers. These manufacturers often turn to simulations to understand how to improve a vehicle's design as real-world safety tests are expensive and time consuming. Understanding the results of these simulations, however, is challenging due to the complexity of the data, which often includes both spatial and nonspatial data types. In this design study we collaborated with analysts who are trying to understand the vulnerability of military vehicles. From this design study we contribute a problem characterization, data abstraction, and task analysis for vehicle vulnerability analysis, as well as a validated and deployed tool called Shotviewer. Shotviewer links 3D spatial views with abstract 2D views to support a broad range of analysis needs. Furthermore, reflection on our design study process elucidates a strategy of view-design parallelism for creating multiview visualizations, as well as four recommendations for conducting design studies in large organizations with sensitive data.

Categories and Subject Descriptors (according to ACM CCS): H.5.2 [Information Interfaces]: User Interfaces—GUI

## 1. Introduction

Simulations enable engineers to test designs in ways that may be too expensive or time consuming for the real world. For example, vehicle manufactures use simulations to understand how a vehicle may perform in extreme conditions, such as during a collision [Hau81], providing insights that can ultimately lead to safer vehicles as engineers revise designs based on the simulation results. Similarly, military or-

ganizations use ballistic simulations to understand the vulnerability of their vehicles, allowing them to improve safety by modifying vehicle designs.

In these military applications, analyzing the vulnerability of vehicles relies heavily on ballistic simulations [Bal03]. Consuming these simulation results, however, requires human insight as analysts must find trends, patterns, and outliers in vehicle vulnerability that require experiential knowledge to spot. Furthermore, the analysts must reason about *multityped data*: output attributes from physics-based simulations, computer-aided design (CAD) models, and hierar-

† kerzner@sci.utah.edu

chical relationships between the model's components. Analysts use their findings to debug simulation inputs, identify components likely to be damaged, plan live-fire testing, and ultimately make recommendations about vehicle design. The complete analysis process is labor intensive and relies heavily on analyst's expert knowledge. As one senior vulnerability analyst told us, consuming ballistic simulation results is "more of an art than a science."

In this design study we worked closely with three vulnerability analysts to demystify the process of consuming ballistic simulation results. The collaboration also enabled us to design and deploy a prototype visualization system that leverages multiple linked views for supporting reasoning about spatial and non-spatial vulnerability data. As a result, several novel contributions arise including a problem characterization, data abstraction, and task analysis for the vulnerability analysis domain, as well as **Shotviewer**, a carefully justified and validated software prototype for visual vulnerability analysis. Furthermore, reflection on our design process illuminates a strategy for exploiting view-design parallelism while creating multiview visualizations, and a list of four recommendations for conducting design studies in large organizations with sensitive data.

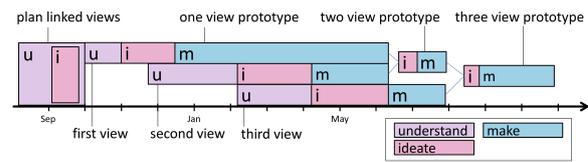
## 2. Methods

We describe our methods using the design activity framework [MMAM14], a process model for visualization design, as it provides the vocabulary necessary to describe the parallel nature of our design process and maps to the well-known nested model for visualization design [Mun09]. The framework identifies four discrete design activities: **understand** the users' needs; **ideate**, or generate ideas, to support those needs; **make** tangible prototypes of the ideas; and **deploy** visualization prototypes to users. We discuss our methods in the context of these activities after describing our project constraints; namely developing remotely, designing for sensitive data, and working within a large organization.

The design study was conducted over the course of fifteen months. We worked remotely throughout the project as we were two time zones away from our collaborators. We met weekly via video conferences, and spent three weeks onsite with analysts. Unless otherwise specified, all the methods were conducted in video conferences, making it challenging to develop rapport as video conferences can be impersonal.

The challenges of working remotely were exacerbated by our collaborators' sensitive data that could not be moved from their secure machines, machines that were not connected to outside networks. We had access to their real data during our onsite visits, but otherwise used simplified test data which is shown in all of the figures in this paper.

Our collaborators are employees of a large organization, which is characterized by specialized individuals relying on cooperation to accomplish tasks coupled with decentralized authority over workflows [SIB11, Dru88]. Thus, we sought



**Figure 2:** The timeline of our design study. After conducting initial understand and ideate activities, we decided on a multiple linked view system. We designed each of these views in parallel and ultimately combined them into a full system.

input from various individuals who fulfill different roles in the organization. In particular, we spoke with engineers who create input data for the analysts and the analysts' customers who communicate results to the vehicle manufacturers.

For the understand activity we initially conducted unstructured interviews with one analyst and one fellow tool builder [SMM12] to learn domain vocabulary and to create an initial data abstraction. We refined the abstractions and performed a task analysis through onsite contextual inquiry [WFB96] with three analysts. To engage the analysts and convince them of the potential for visualization to improve their workflow, we conducted three visualization awareness workshops by following Koh and Slingsby's user-centered model [KS11].

For both the ideate and make activities we used parallel prototyping [DGK\*10] and participatory design [MK93], but with prototypes of varying fidelity for the two activities. For instance, we used low-fidelity paper prototypes during the ideate activity compared to higher fidelity data sketches [LD11] in the make activity. To elicit feedback about our prototypes, we used the *rose-bud-thorn method* [Ins14], where individuals are prompted to give three comments on an idea: one positive (rose), one negative (thorn), and one identifying an opportunity for future work (bud). This allowed us to elicit both positive and negative feedback even though our collaborators were initially reluctant to criticize our ideas.

We deployed our final visualization system, Shotviewer, to three analysts who have used it in their daily work. Wide-scale deployment, however, requires integrating it with existing vulnerability analysis software. We discuss the proposed integration in Section 5 and it is currently being implemented by developers at our collaborator's organization.

Although we discuss the design activities linearly here, we performed them in a parallel and staggered fashion as shown by our project's timeline in Figure 2. Our software design consists of three linked views to display spatial and nonspatial data, and each row in the timeline corresponds to per-view activities. For simplicity, we do not show the iterative nature of the design process which often involves backward movement through activities. Section 7 contains



**Figure 3:** A cell plot output from ballistic simulations. Cell color encodes the quantitative damage that each shot inflicts on this vehicle's mobility or firepower capabilities. Analysts understand cell plots by comparing cells to their neighbors of different colors, such as the highlighted cells marked here.

our strategy and justification for designing multiple linked views in parallel.

### 3. Problem, Data, and Tasks

In this section we discuss our problem characterization, data abstraction, and task analysis contributions for the domain of ballistic vulnerability analysis. First we characterize the domain problem by providing a high-level overview of ballistic simulations and describing the broad goals of vulnerability analysts. Next we propose a data abstraction for the simulation, followed by the specific tasks the analysts need to conduct in order to understand the simulation results.

#### 3.1. Problem Characterization

Our collaborators focus on understanding ballistic simulation outputs. These outputs are best explained by examining the origin of ballistic simulation software: an optical ray tracer [Mat67, BS07]. Ray tracers compute photon paths through an environment, shade surfaces using physically-based lighting models, and output pixel color based on primary visibility rays. Ballistic simulations replace photons with **shots**, a projectile being simulated, and compute energy transfer using physically-based penetration models. Historically, the simulations output statistical summaries such as the total vulnerable area of a vehicle, but these summaries have a relatively nebulous definition that make it almost impossible to reason about why a vehicle may be vulnerable.

To understand simulation outputs in greater detail, analysts rely on images output from the ballistic simulations, called **cell plots** as shown in Figure 3, where cell color represents vehicle capability damage from a shot for a given trajectory. The number of cell plots and cell size vary between vehicles and analyses. Typically, analysts run simulations for between three and forty-two industry-standard trajectories around a vehicle, with a cell size ranging from 10-100mm [Bal03]. Unfortunately, consuming cell plots is labor intensive as they are ambiguous: they contain no information about *why* a cell is a certain color. These colors are derived from a variety of information, both spatial and non-spatial, related to a vehicle's 3D geometry and capabilities.

Vulnerability analysts rely on their extensive domain knowledge to intuit about the cause of a cell's color, which they use to identify patterns and outliers in vehicle vulnerability.

#### 3.2. Data Abstraction

Figure 4 contains an overview of the simulation data relevant to understanding cell plots. The vehicle inputs are collections of 3D meshes, called **components**, shown in Figure 4(a). A **dependency graph**, shown in Figure 4(b) describes the functionality of the vehicle in terms of its components [VGRH81]. The graph's leaves are **critical components**, a subset of the vehicle's components that directly contribute to its capabilities. The graph's internal nodes are aggregations of components called **systems**, and its roots are aggregations of systems called **capabilities**.

Figure 4(c) shows the simulation launching a shot at the vehicle. For a given trajectory, the simulations compute **shotlines**, the path of a shot through the vehicle. These shotlines contain the physical properties of the shot — quantitative attributes such as mass and velocity defined along the 1D line. Shotlines also contain per-component quantitative values that represent the amount of damage to intersected components [Bal03]. Simulations aggregate per-component damage up the dependency graph from components to capabilities in order to compute per-shotline capability damage values. Each cell in a cell plot, which represents a single shotline, is then colormapped to encode the damage value.

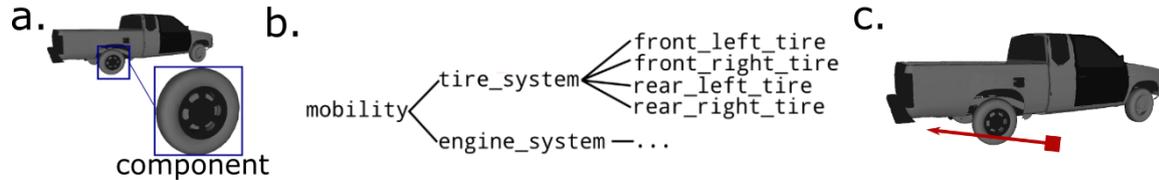
These damage values, and resulting cell plots, are an ineffective representation of the simulation output as they aggregate complex spatial and abstract data into a single value. We focus on unpacking and visualizing the information behind cell plots in order to understand the rich and descriptive simulation data.

#### 3.3. Task Analysis

The high level goal of this work is to enable a deeper understanding of the data behind cell plots. Analysts use cell plots to debug simulation inputs and to understand vehicle vulnerability. In both cases, they must explain the color behind an individual cell or a group of cells to check the validity of their inputs or to identify trends in vulnerability. Our task analysis revealed that analysts understand cells in the context of their spatially adjacent neighbors. For instance, the differences between red and white cells highlighted in Figure 3 can be used to explain what components are being damaged by the central group of red cells. To understand differences between shots, analysts perform the following tasks per-shot and compare their outcomes:

- T1 Understand a shot's degradation and component damage.
- T2 Understand a shot's spatial context.
- T3 Understand a shot's systemic impact.

These three tasks require understanding the simulation output, i.e., shots and component damage, in the context



**Figure 4:** An overview of ballistic vulnerability analysis data: a) vehicles consist of various 3D meshes called components; b) a subset of these components appear in a dependency graph which describes the vehicle's capabilities; and c) the simulations trace shots through the vehicle and aggregate per-shot damage using the dependency graph.

of the simulation input, i.e., the vehicle's geometry and dependency graph — we describe these tasks in greater detail through our design requirements in Section 5. Currently, analysts understand a shot's degradation and component damage (T1) using textual debugging logs. These logs are massive and not particularly human readable — in our test data one of the debugging logs is 2.7Gb in size. Furthermore, analysts trace a shot's systemic impact (T3) by viewing yet another textual description of the dependency graph. Finally, to see a shot's spatial context (T2) the analysts rely on offline rendering software or their own mental models. We postulate that effectively making sense of this multityped simulation data requires a multiview visualization that combines both the abstract 2D and the spatial 3D data.

#### 4. Related Work

To our knowledge, no previous research has focused on visualizing ballistic simulation results. Recent work within the vulnerability analysis domain involves accelerating simulations with modern ray tracers [BS07] and designing software for preparing simulation inputs [SUR13]. Ours is the first project to analyze the consumption of simulation results. Due to the lack of existing visualization software for vulnerability analysis, in this section we discuss the research related to the challenges of our design study, specifically: creating 2D representations of inherently spatial data and designing multiview visualizations for multityped data.

Visual comparison of inherently 3D data is made challenging by occlusion, clutter, and difficult navigation. To avoid these issues, we project 3D geometry and shot data into 2D while preserving spatial relationships. This is inspired by Keefe et al. [KERC09], who derive 2D representations of 4D animations with geometry tracers. Similarly, Landge et al. [LLI\*12] project 3D network structures to a 2D view in order to avoid occlusion while preserving spatial relationships. Also, Weber et al. [WRH\*09] and Meyer et al. [MMDP10] represent embryo cell position using a lossless 2D parameterization of its structure. Recently, Al-Awami et al. [AABS\*14] propose a 2D representation of brain connections that preserves spatial relationships. Similar to these designs, we use a 2D projection of 3D geometry and shot data.

We also use linked multiform 2D and 3D views. Closely related to our work is the SimVis application [DGH03] as it

combines spatial 3D and abstract 2D views, though the 2D representations are limited to scatterplots and histograms. Similarly, the GRACE application [MMH\*13] combines 3D spatial brain data with abstract functionality data, but does not consider 2D projections of the 3D data. Chang and Collins [CC13] augment 3D views with 2D views containing summaries of vehicle damage from highway reports, but they do not consider the resulting vehicle functionality from that damage. Design studies on in-car communication networks [SBH\*09, SFMB12] also combine 2D abstract views with 3D spatial views, though they do not apply to the problem of vulnerability analysis.

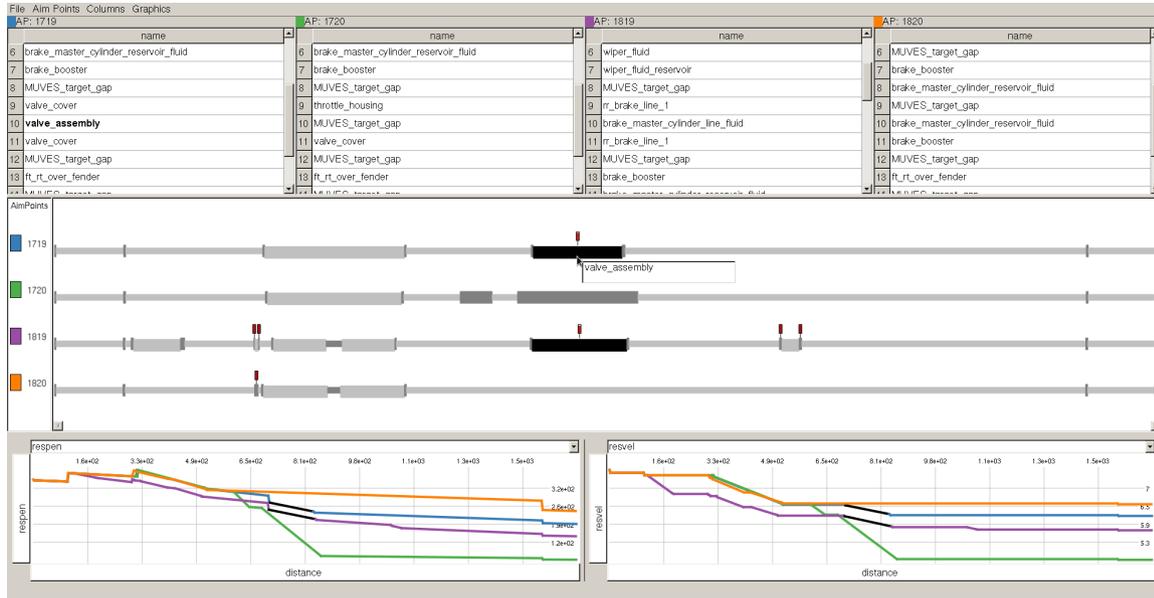
#### 5. Shotviewer

In this section we present Shotviewer, a prototype visualization system that combines spatial and nonspatial data for ballistic vulnerability analysis. Its three linked views, inspired by the multiform encodings of Maries et al. [MMH\*13], correspond to the tasks of understanding cell plots: the **Shotline View** enables comparison of shot degradation and component damage (T1); the **Geometry View** provides information about a shot's spatial context (T2); and the **System View** displays a shot's systemic impact (T3). We have released Shotviewer's source code and a sample dataset at: <http://www.ethankerzner.com/projects/shotviewer>

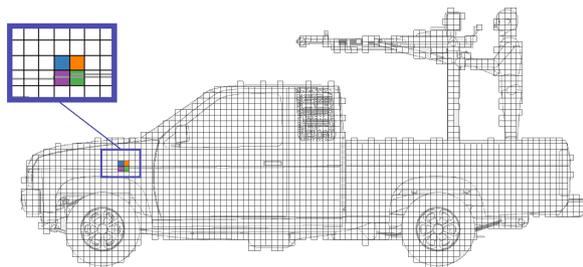
We discuss Shotviewer by identifying per-view requirements and using them to justify our design decisions. We preface this with two application-wide requirements:

*Support current workflow and offer new capabilities.* Analysts read text files to make sense of cell plots, allowing them to compare two shotlines at a time. We created Shotviewer to visualize from one to four shotlines simultaneously. Although Shotviewer will not scale to more than four shotlines, it supports analysts current workflow and offers new capabilities [SIBB10] that we discuss for each view. Shotviewer also incorporates textual data representations where possible to help analysts build trust in the visualizations as they can verify that the encodings match the data.

*Interface with existing tools.* We designed Shotviewer so that analysts may launch it from their existing software. Existing software [SUR13] displays cell plots to which Shotviewer then provides details-on-demand. We propose that cell plots act as a legend by assigning a categorical color



**Figure 6:** The Shotline View consists of three subviews. The Table View (top) provides human-readable details about the shot. The Compare View (center) uses a 2D projection of shotlines to enable visual comparison through juxtaposition [GAW\* 11]. The two Line Plots Views (bottom) show trends in shot degradation.



**Figure 5:** We propose that cell plots serve as an interactive legend for Shotviewer. Here, the user selected four shotlines which have been assigned categorical colors that are used within Shotviewer.

to each cell, as shown in Figure 5. We use these colors to identify shots within our application.

### 5.1. Shotline View

The Shotline View in Figure 6 shows information about the shots' degradation and the damage to intersected components. This information is shown in three subviews: a Table View (top); a Compare View (center); and Line Plot Views (bottom). While the Table View supports existing workflows with textual data representations, we based the other subviews on the following requirements:

*Show shotlines as linear events.* The Compare View uses a lossless 1D parameterization of shotlines. The shot's vehicle entry point is at the far left and horizontal position encodes distance from it. We represent shots with straight lines as penetration models do not allow for refraction. This linear

display corresponds to reading textual shot descriptions from the entry point through the vehicle.

*Differentiate between air and components, while preserving thickness of arbitrarily sized components.* We represent air and components along a shotline using a horizontal set of rectangles. Since analysts focus on component damage, rectangles representing components have larger height than those representing air. Rectangle width encodes the component thickness. This is particularly useful as component thickness is important in physically-based ballistic models. Rectangle color has no inherent meaning, but rather alternates to distinguish between components. This is necessitated by the varying width of components, for instance, ranging from an engine block to an electrical wire. Our parameterization and damage representations were inspired by the linear gene representation in Variant View [FNM13].

*Display damaged components.* The Compare View shows component damage using the height and color of a fixed width glyph above each component. We set the width to a fixed size to show damage of small and large components with equal salience. This encoding is particularly useful in vehicles with complex electrical wiring, which is common in our collaborators' sensitive data.

*Enable comparison of shot degradation between and within shotlines.* Shot degradation is the change in mass, velocity, and other quantitative values defined on the shotline. We initially considered encoding these in the Compare View, for instance, by using rectangle height, but with such encodings it is difficult to compare values between shotlines. We

instead use the two Line Plot Views to display the shots' physical degradation: the x-axis uses the same lossless parameterization as the Compare View and the y-axis encodes the physical properties. These views enable comparison both along and between shotlines, allowing analysts to verify that the physically-based simulations work as intended.

*Identify components hit by multiple shots.* Interaction allows analysts to highlight components that have been shot, which is useful for identifying components hit by more than one shot, such as the valve\_assembly in Figure 6 which has been hit by two shots. Moving the mouse over a component in any of these sub-views highlights it throughout the application in order to support reasoning about the component's spatial location and systemic impact.

## 5.2. Geometry View

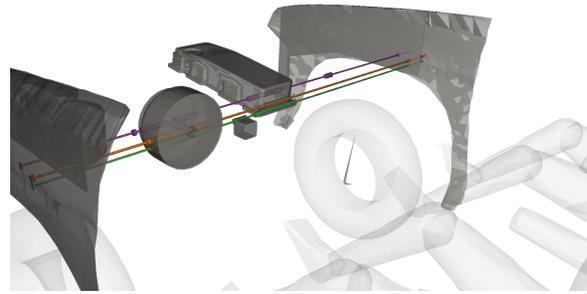
The Geometry View in Figure 7 displays shots' 3D position in the context of the vehicle geometry. We designed this view from the following requirements:

*Represent shots and shot components in 3D.* The Geometry View displays shotlines as 3D cylinders. Similar to the Shotline View it uses colors to identify shots and cylinder radius to differentiate between air and components. It also renders each of the components intersected by the shots.

*Show spatial context with respect to vehicle geometry.* Rendering just the shotlines and intersected components provides insufficient spatial context about the entire vehicle. Rendering the entire vehicle, however, is impractical due to slow rendering performance and visual clutter. While we initially considered exploded geometry views [LACS08], they failed to provide information about the true spatial relationships of components needed by our analysts. Instead, we observed the analysts using offline renderers to draw intersected components along with the geometry of known locations such as crew or wheels in order to place the components in their mental model of the vehicle. Based on this observation we developed the concept of **landmark geometry**, a user-defined set of recognizable components. In Shotviewer we allow analysts to define landmark geometry with regular expressions.

*Display spatial context with respect to specific systems.* Aliasing leads to unexpected simulation results as shotlines simulated by infinitely thin rays fail to intersect high-frequency geometry, such as wires. Often, analysts will verify that aliasing occurred by looking at a shot with respect to an individual system they expect to be damaged. We support this reasoning by letting users select systems rendered with the shotlines, shot components, and landmark geometry.

*Customize colors used in 3D rendering.* In their existing tool chains, analysts have little control over color schemes used in 3D rendering. As one analyst complained, the vehicles "look like clown cars." The analysts would often manually color geometry images when presenting results to their



**Figure 7:** The Geometry View renders 3D representations of the shotlines along with user-defined landmark geometry to provide spatial context while avoiding clutter and occlusion. Here, the crew and wheels serve as landmark geometry.

peers or customers. Thus, in Shotviewer we allow users to control the geometry color and opacity. Although this feature may seem insignificant, improving the visualization aesthetics significantly improved our rapport with analysts.

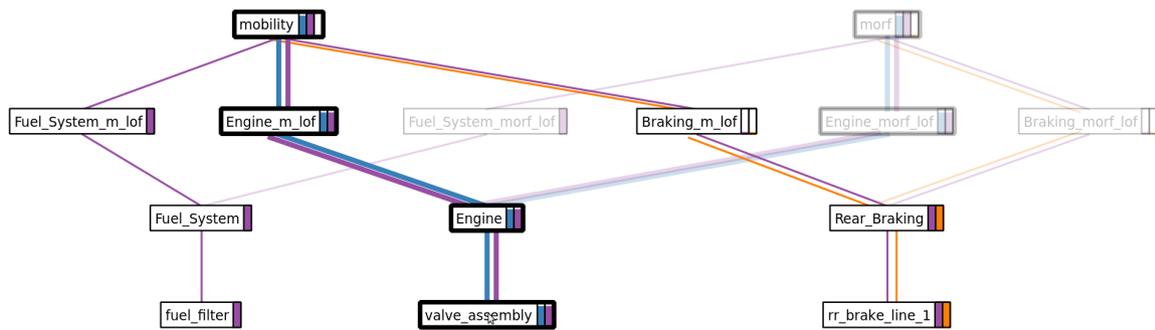
## 5.3. System View

The System View in Figure 8 visualizes systemic impact of component damage using a node-link diagram. Although the simulations output a list of damaged components, these lists fail to sufficiently describe the impact of that damage. For example, in Figure 8 the valve\_assembly has been damaged and it would be difficult to intuit that this is a part of the engine system without the dependency graph for context. The idea of showing the dependency graph structure appears in our design requirements for the System View:

*Show only relevant components, systems, and capabilities.* Dependency graphs often contain up to thousands of nodes. While we initially considered space-efficient representations of these graphs, such as treemaps [JS91], analysts found these hard to understand. We instead filter the dependency graph to only nodes impacted by selected shots.

*Clearly encode parent-child relationships and damage.* Analysts must understand the damage propagation through a dependency graph, from the components (leaves) through the systems (internal nodes), and to the capabilities (roots). We use a hierarchical node-link diagram layout (computed with graphviz [GN00]) as it encodes parent-child relationships both with connections and spatial position. We display damage with the height of a vertical bar to the right of each node, and we identify the shot causing damage with the bar's color. Links represent damage propagating up the tree. Moving the mouse over a node highlights the path to its parents.

*Enable top-down analysis (from capabilities to components).* Before our design study, analysts could only understand systemic impact by aggregating component damage up to the capabilities. In essence, manually aggregating the details to get an overview of the data. In Shotviewer, we additionally allow analysts to select a node of interest and filter



**Figure 8:** The System View uses a node-link diagram to show the systemic impact of damaged components. The full dependency graph is filtered to only show damaged components (leaves) and their parents (systems and capabilities). The vertical bars on the right of each node encode damage, and color identifies the shots causing the damage. Using this view analysts can filter the damaged capabilities to focus on one of interest. Here, highlighting enables the user to trace the valve\_assembly damage up to the vehicle's mobility.

the graph from the top-down, showing only damaged nodes that contribute to their selection. This top-down workflow is novel in the vulnerability analysis domain.

#### 5.4. Implementation

We implemented Shotviewer using C++ and Qt. We focused on rapid prototyping in order to quickly get user feedback and iterate on designs. We also considered the following requirement in our implementation:

*Develop where we deploy.* We treated the hardware and software available to analysts as constraints. For instance, analysts did not have graphics-processing units, so we used a CPU ray tracing library [WWB\*14] to render the 3D geometry. We also found that virtual machines did not reproduce the behavior of analysts' machines (for example, with windowing libraries that use hardware-accelerated rendering) so we instead developed on real machines that matched their configurations. This enabled fast installation and feedback of our prototypes.

#### 6. Validation

We validate both our task analysis and Shotviewer's design. Early in the project, we interviewed our analysts' customers to find immediate validation on our choice of understanding cell plots. After completing the design study, we sought informal user-feedback from three vulnerability analysts and two case studies that describe Shotviewer's usefulness.

Talking to our analysts' customers provided initial validation of our problem characterization and task analysis. Through early interviews we began to understand the importance of cell plots as well as how difficult they are to interpret, leading to our task analysis that focuses on understanding these plots. One of the analysts' customers who is responsible for communicating vulnerability analysis results to vehicle manufacturers validated this analysis. In particular, he stated "we don't actually do a good job of turning

cell plots into actionable knowledge" and that "the data behind cell plots are intelligible to only a few people." Creating software to understand cell plots, he confirmed, would be beneficial to the entire vulnerability analysis community.

We received overwhelmingly positive user feedback on Shotviewer. One analyst stated that the tool reduced his time-to-understanding the data behind cell plots from minutes to seconds. We have also seen a growing demand for Shotviewer as our analysts' coworkers have seen them working with it and asked to use it for their own work.

Our first case study occurred while demonstrating a final version of Shotviewer to an analyst. She was examining a dataset from a project that she already completed and shipped to a customer. Although she believed the data to be bug free, Shotviewer enabled her to discover an error in it. Particularly, while looking at a shot with a low capability damage, she noticed a component in the Geometry View that was hit by the shot. Although she expected that component to cause a high capability damage, it did not appear in the System View. After checking the simulation inputs, she discovered an error that caused damage to that component to be incorrectly aggregated.

We present a second case study from an analyst who used Shotviewer to understand simulations of a new shot type against an existing vehicle. After preparing inputs and running the simulation, she inspected a cell plot. The cell plot contained some groups of red cells (high capability damage) surrounding a green cell (low capability damage). She launched Shotviewer to see the data behind these cells so that she could explain their differences. Using the Compare View's damage glyphs and interactive highlighting, she identified a component that was damaged by the three red shotlines but not by the green one. She hypothesized that this component caused the high capability damage and used the System View to confirm this by tracing the path from the leaf up through its root. Using the Geometry View, she was

also able to see that the green shotline had passed very close to the component and that it would have likely been damaged in a live-fire test. She concluded that the outlier was a result of aliasing and the component in question was vulnerable to this particular shot. In her old work flow, this would have involved looking up information from three different text files and running an offline rendering application to see the shots' spatial context.

## 7. Reflections

In this section we present the two methodological contributions that result from reflection on our design process. The first is a strategy that exploits **view-design parallelism** for creating multiview visualizations of multityped data. This strategy proposes designing views in a parallel and staggered fashion in order to effectively use time and resources while keeping collaborators engaged in the design process. The second contribution is a list of four recommendations to increase the efficiency of designing visualizations in a large organization with sensitive data.

### 7.1. View-Design Parallelism

View-design parallelism complements the growing area of design study process models [SMM12, GDJ\*13, WBD14, MMAM14]. While these models focus on the iterative and multilinear nature of design, including the execution of parallel design activities, they do not directly address the challenges that we as designers face in problem driven work: potentially overwhelming amounts of data and collaborators who expect tangible prototypes early on in the design process. View-design parallelism is a divide-and-conquer strategy for effectively designing individual views of a multiview visualization system. We present this strategy in the context of our work with Shotviewer, where it enabled us to avoid paralysis caused by staggering amounts of data and uncountably many design possibilities. It allowed us to quickly deliver tangible prototypes which refined our understanding of the problem and further engaged our collaborators.

The need for multiview visualizations arises when a single view is not optimal for all necessary tasks [Mun15], such as with ballistic simulation data. In the case of Shotviewer, we knew early on that we would need multiple views to support the three different types of data. At first we were overwhelmed while trying to understand all of the data and associated tasks at once. We noticed that by trying to stick to the understand activity for all three data types, our efforts turned counter productive as we, and our collaborators, became mentally exhausted. To overcome this, we pushed on to the ideate and make activities for one of the data types, while still understanding the other two. This injected new energy into the project, gave us tangible results to pass onto our collaborators, and also helped us to better understand the nuances of the data and tasks overall.

We continued to design the three views in a staggered and parallel fashion, a strategy we call view-design parallelism.

The key idea of view-design parallelism is to begin with the understand activity for one data type, and then when moving onto the ideate activity, beginning the understand activity for a second data type, and so on — this continues for each data type and individual view. This strategy is shown in our timeline in Figure 2. However, this strategy leaves open the question of how to decide which data type and view to focus on first, second, etc. To answer this, we advocate the use of the following considerations to rank each data type:

1. Availability: do we have the necessary data and support?
2. Usefulness: will our collaborators use the visualization?
3. Impact: will the visualization impact our collaborators?
4. Time: how quickly can we deliver the visualization?

The first item is a constraint: it is impossible to design views without data and support such as parsers [SMM12]. The other items are considerations that may vary between projects. In essence, when we have the opportunity to begin designing a view, we select one that is useful, that will have a large impact, and that can be designed quickly.

While building Shotviewer we exploited view-design parallelism. We started with the Shotline View as we had access to the shotline data while waiting on access to parsers for the other data types. Next, we built the System View as we believed it would have a significant impact by enabling a new way of looking at the dependency graph through top-down analysis. We concluded by building the Geometry View as it was the most labor intensive in terms of implementation. Throughout our design process we were able to transfer insights between the design of each view. For instance, in designing the System View we used the encoding of vertical height to show per-component damage values that we had already designed and tested for the Shotline View.

After designing the individual views we began to think about how to combine them into a multiview system. At this point, we benefited from getting user feedback on the views individually, as well as connected to each other. Through informal experiments we observed that presenting just single views to users elicited feedback on the individual encodings, while presenting views linked together elicited feedback on the choice of interaction and linking. This observation helped us decide which views to show users in order to focus feedback appropriately.

In summary, view-design parallelism is a strategy for designing visualizations that overcomes the challenges of understanding overwhelming data, of quickly creating tangible prototypes, and of diminishing returns for time spent on individual design activities. We believe it is particularly useful when designing multiview visualizations. A useful metaphor to describe the strategy is to compare visualization design to a reduced instruction set CPU architecture that exploits instruction-level parallelism [HP11]. The design of each individual view is like a CPU instruction — just as CPU pipelines overlap and stagger instruction execution to efficiently use limited hardware resources (e.g., for instruction

fetch, decode, arithmetic, and memory access), view-design parallelism overcomes our limited mental resources (e.g., time and energy dedicated to a single activity). Instruction-level parallelism increases CPU throughput, and similarly we believe view-design parallelism increases productivity of visualization design.

## 7.2. Recommendations

Data constraints have become increasingly common with problem-driven research — in our case, these constraints arose from working on sensitive data within a large organization. While researchers are sometimes able to obtain surrogate data with similar characteristics to the real data, such as Walker et al. [WSD\*13] who used online business reviews in place of human-terrain reports, there are domains where no such surrogate data exist, such as in vulnerability analysis. In other cases metadata can be used to create visualizations without direct data access [MEV14], but metadata does not exist for our domain. Thus, we were forced to work with simplified test data, and we identify the following recommendations based on our experience. These recommendations are an extension of previously identified design study pitfalls [SMM12] and recommendations for evaluating visualizations in large organizations [SIBB10].

*Sample the relevant data pipeline.* By talking with the *producers* and *consumers* of our analysts' data, we built a more accurate data abstraction. These conversations also allowed us to validate our initial choice of explaining data behind cell plots early in the design process.

*Recognize test data is not real data.* By talking with our collaborators about how our test data differed from the real data we were able to develop visualizations that were more likely to work on the real data, for instance, by handling high-frequency geometry in the Shotline View.

*Budget time for transitioning to real data.* During our onsite visits we realized that the real data did not always match the format of our test data. By budgeting time at the beginning of our onsite visits specifically for debugging our application with real data, we were able to ensure we could demo a working system to our collaborators with their real data, maximizing our productivity during the limited time we could spend with them.

*Automate everything.* By automating the installation process of Shotviewer we were able to more quickly get feedback from analysts on new designs. This was particularly useful when we could not remotely access our analysts machines due to their data's sensitive nature.

Although these four recommendations appear obvious in hindsight, they are solutions to pitfalls that we encountered throughout our project. We believe they are worthwhile considerations for future design studies conducted within large organizations and with sensitive data.

## 8. Conclusion and Future Work

In this paper we present the results of a fifteen month design study in the domain of vulnerability analysis. The contributions of this work include a problem characterization, data abstraction, and task analysis for this domain, as well as Shotviewer, a prototype vulnerability visualization tool that uses multiple linked views to display spatial and nonspatial data. We validate Shotviewer with user feedback and two case studies. Reflections on our design process also present two methodological contributions: view-design parallelism, a strategy for designing multiview visualizations; and four recommendations for conducting design studies in large organizations with sensitive data.

This design study focuses on effective encodings for multiview visualizations of multityped data. Future work could formally describe and evaluate user-interactions with multiview systems. We are also interested refining the process of problem-driven research in the context of large organizations, in particular by identifying a methodology for balancing research with product development.

Our design ultimately addresses one small piece of the vulnerability analysis pipeline: explaining the results of individual simulations. In the future, we hope to use visualization to aggregate and summarize large numbers of ballistic simulation results.

## Acknowledgments

We acknowledge our collaborators at the US Army Research Laboratory and affiliated organizations: Matthew Rothwell, Carolyn Stancoff, Rick zum Brunnen, Cletus Hunt, James Hunt, Jefferson Amstutz, Scott Shaw, Christiaan Gribble, and Mark Butkiewicz. We thank Jodi Robertson and Susan Coates for providing us with releasable data. Nathan Galli helped create our video. We also recognize Alex Bigelow, Sean McKenna, Nina McCurdy, Sam Quinan, and Kris Zygmunt for their feedback on this work.

## References

- [AABS\*14] AL-AWAMI A. K., BEYER J., STROBELT H., KASTHURI N., LICHTMAN J. W., PFISTER H., HADWIGER M.: NeuroLines: A Subway Map Metaphor for Visualizing Nanoscale Neuronal Connectivity. *IEEE Trans. on Vis. and Comp. Graphics* 20, 12 (2014).
- [Bal03] BALL R. E.: *The Fundamentals of Aircraft Combat Survivability Analysis and Design*. American Institute of Aeronautics and Astronautics, 2003.
- [BS07] BUTLER L. A., STEPHENS A.: Bullet Ray Vision. In *IEEE Symp. on Interactive Ray Tracing* (2007).
- [CC13] CHANG M.-W., COLLINS C.: Exploring Entities in Text with Descriptive Non-photorealistic Rendering. In *IEEE Pacific Vis. Symp.* (2013).
- [DGH03] DOLEISCH H., GASSER M., HAUSER H.: Interactive Feature Specification for Focus + Context Visualization of Complex Simulation Data. In *Symp. on Data Vis.* (2003).
- [DGK\*10] DOW S. P., GLASSCO A., KASS J., SCHWARZ M., SCHWARTZ D. L., KLEMMER S. R.: Parallel Prototyping Leads

- to Better Design Results, More Divergence, and Increased Self-Efficacy. *ACM Trans. on Comp.-Human Interaction* 17, 4 (2010).
- [Dru88] DRUCKER P.: The Coming of the New Organization. *Harvard Business Review*, January (1988).
- [FNM13] FERSTAY J. A., NIELSEN C. B., MUNZNER T.: Variant View: Visualizing Sequence Variants in Their Gene Context. *IEEE Trans. on Vis. and Comp. Graphics* 19, 12 (2013).
- [GAW\*11] GLEICHER M., ALBERS D., WALKER R., JUSUFI I., HANSEN C. D., ROBERTS J. C.: Visual Comparison for Information Visualization. *Information Visualization* 10, 4 (2011).
- [GDJ\*13] GOODWIN S., DYKES J., JONES S., DILLINGHAM I., DOVE G., ALLISON D., KACHKAEV A., SLINGSBY A., WOOD J.: Creative User-Centered Visualization Design for Energy Analysts and Modelers. *IEEE Trans. on Vis. and Comp. Graphics* 19, 12 (2013).
- [GN00] GANSNER E. R., NORTH S. C.: An Open Graph Visualization System and Its Applications to Software Engineering. *SOFTWARE - PRACTICE AND EXPERIENCE* 30, 11 (2000).
- [Hau81] HAUG E.: Engineering Safety Analysts vis Destructive Numerical Experiments. *EUROMECH 121, Polish Academy of Sciences, Engineering Trans.* 29, 1 (1981).
- [HP11] HENNESSY J. L., PATTERSON D. A.: *Comp. Architecture: A Quantitative Approach*. Morgan Kaufmann, 2011.
- [Ins14] INSTITUTE L.: Vision Statement: A Taxonomy of Innovation, 2014.
- [JS91] JOHNSON B., SHNEIDERMAN B.: Treemaps: A Space-Filling Approach to the Visualization of Hierarchical Information Structures. In *IEEE Conf. on Visualization* (1991).
- [KERC09] KEEFE D. F., EWERT M., RIBARSKY W., CHANG R.: Interactive Coordinated Multiple-View Visualization of Biomechanical Motion Data. *IEEE Trans. on Vis. and Comp. Graphics* 15, 6 (2009).
- [KS11] KOH L., SLINGSBY A.: Developing and Applying a User-centered Model for the Design and Implementation of Information Visualization Tools. In *IEEE Conf. Information Visualisation* (2011).
- [LACS08] LI W., AGRAWALA M., CURLESS B., SALESIN D.: Automated Generation of Interactive 3D Exploded View Diagrams. In *Proc. of SIGGRAPH* (2008).
- [LD11] LLOYD D., DYKES J.: Human-Centered Approaches in Geovisualization Design: Investigating Multiple Methods Through a Long-Term Case Study. *IEEE Trans. on Vis. and Comp. Graphics* 17, 12 (2011).
- [LLI\*12] LANDGE A. G., LEVINE J. A., ISAACS K. E., BHATELE A., GAMBLIN T., SCHULZ M., LANGER S. H., BREMER P.-T., PASCUCCI V.: Visualizing Network Traffic to Understand the Performance of Massively Parallel Simulations. *IEEE Trans. on Vis. and Comp. Graphics* 8, 12 (2012).
- [Mat67] MATHEMATICAL APPLICATIONS GROUP INC.: *A Geometric Description Technique Suitable for Comp. Analysis of Both Nuclear and Conventional Vulnerability of Armored Military Vehicles*. Tech. rep., 1967.
- [MEV14] MADHAVAN K., ELMQVIST N., VORVOREANU M.: DIA2: Web-based Cyberinfrastructure for Visual Analysis of Funding Portfolios. *IEEE Trans. on Vis. and Comp. Graphics* 20, 1 (2014).
- [MK93] MULLER M. J., KUHN S.: Participatory Design. *Communications of the ACM* 36, 6 (1993).
- [MMAM14] MCKENNA S., MAZUR D., AGUTTER J., MEYER M.: Design Activity Framework for Visualization Design. *IEEE Trans. on Vis. and Comp. Graphics* 20, 12 (2014).
- [MMDP10] MEYER M., MUNZNER T., DEPACE A., PFISTER H.: MulteeSum: a Tool for Comparative Spatial and Temporal Gene Expression Data. *IEEE Trans. on Vis. and Comp. Graphics* 16, 6 (2010).
- [MMH\*13] MARIES A., MAYS N., HUNT M., WONG K. F., LAYTON W., BOUDREAU R., ROSANO C., MARAI G. E.: GRACE: A Visual Comparison Framework for Integrated Spatial and Non-Spatial Geriatric Data. *IEEE Trans. on Vis. and Comp. Graphics* 19, 12 (2013).
- [Mun09] MUNZNER T.: A Nested Model for Visualization Design and Validation. *IEEE Trans. on Vis. and Comp. Graphics* 15, 6 (2009).
- [Mun15] MUNZNER T.: *Visualization Analysis and Design*. CRC Press, 2015.
- [SBH\*09] SEDLMAIR M., BERNHOLD C., HERRSCHER D., BORING S., BUTZ A.: MostVis: An Interactive Visualization Supporting Automotive Engineers in MOST Catalog Exploration. In *IEEE Conf. Information Visualisation* (2009).
- [SFMB12] SEDLMAIR M., FRANK A., MUNZNER T., BUTZ A.: RelEX: Visualization for Actively Changing Overlay Network Specifications. *IEEE Trans. on Vis. and Comp. Graphics* 18, 12 (2012).
- [SIB11] SEDLMAIR M., ISENBERG P., BAUR D.: Cardiogram: Visual Analytics for Automotive Engineers. In *Proc. of SIGCHI* (2011).
- [SIBB10] SEDLMAIR M., ISENBERG P., BAUR D., BUTZ A.: Evaluating Information Visualization in Large Companies: Challenges, Experiences and Recommendations. In *Workshop on Beyond Time and Errors: Novel Evaluation Methods for Information Visualization (BELIV)* (2010).
- [SMM12] SEDLMAIR M., MEYER M., MUNZNER T.: Design Study Methodology: Reflections from the Trenches and the Stacks. *IEEE Trans. on Vis. and Comp. Graphics* 18, 12 (2012).
- [SUR13] SURVICE ENGINEERING: The Advanced Joint Effectiveness Model. In <http://ajem.com/>. 2013.
- [VGRH81] VESELY W. E., GOLDBERG F. F., ROBERTS N. H., HAASL D. F.: *Fault Tree Handbook*. Tech. rep., U.S. Nuclear Regulatory Commission, Washington, D.C., 1981.
- [WBD14] WOOD J., BEECHAM R., DYKES J.: Moving beyond sequential design : Reflections on a rich multi-channel approach to data visualization. *IEEE Trans. on Vis. and Comp. Graphics* 20, 12 (2014).
- [WFB96] WIXON D., FLANDERS A., BEABES M. A.: Contextual Inquiry: Grounding Your Design in User's Work. In *CHI '96 Conf. on Human Factors in Computing Systems* (1996).
- [WRH\*09] WEBER G. H., RÜBEL O., HUANG M.-Y., DEPACE A. H., FOWLKES C. C., KERÄNEN S. V. E., LUENGO HENDRIKS C. L., HAGEN H., KNOWLES D. W., MALIK J., BIGGIN M. D., HAMANN B.: Visual Exploration of Three-Dimensional Gene Expression using Physical Views and Linked Abstract Views. *IEEE/ACM Trans. on Computational Biology and Bioinformatics* 6, 2 (2009).
- [WSD\*13] WALKER R., SLINGSBY A., DYKES J., XU K., WOOD J., NGUYEN P. H., STEPHENS D., WONG B. L. W., ZHENG Y.: An Extensible Framework for Provenance in Human Terrain Visual Analytics. *IEEE Trans. on Vis. and Comp. Graphics* 19, 12 (2013).
- [WWB\*14] WALD I., WOOP S., BENTHIN C., JOHNSON G. S., ERNST M.: Embree: A Kernel Framework For Efficient CPU Ray Tracing. *ACM Trans. on Graphics* 33, 4 (2014).