Improving Parameterized Design with Interactive User-Guided Sampling and Parameter Identification Tools

Evan Shimizu

CMU-CS-20-104

August 21, 2020

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was sponsored by a gift from Facebook, Inc.; gifts from Google LLC; a gift from Lutron Electronics Company; a gift from Nvidia Corporation; gifts from Oculus VR; and a gift from Vodafone Group.

Keywords: Creativity support tools; design tools; parameterized design; sampling methods; design interfaces; lighting design; image editing

Dedicated to all of the artists and designers who have asked "why can't we have nice things" while using a computer.

Abstract

Modern computer graphics design tasks often take place in high-dimensional parameterized design spaces. In these spaces, the design is specified by the value of tens to hundreds of parameters which often interact in ways that are difficult to predict. For instance, a parametric font may have tens of parameters controlling everything from stroke thickness to serif appearance, while a parameterized material may have hundreds of parameters specific to the material, such as a brick material providing controls for number of bricks per row and column, but no single brick size parameter. In these parameterized domains, creating a design often follows a coarse-to-fine iterative process where a designer creates a set of initial designs that are gradually refined until a design meeting all constraints is created. Per-parameter interfaces commonly used for parameterized design are not well-aligned with this process. One common method of providing higher-level navigation is to enable visual exploration by using a design gallery. This gallery presents an organized collection of samples selected from the design space that the user can browse through. Gallery interfaces provide a solid overview of a design space, but are difficult to direct to specific regions based on the user's current design goal.

This thesis presents a collection of software systems and interface techniques that support productive design in high-dimensional parameter spaces through interactive user-guided sampling. A major component of this work is Design Adjectives, a domain-agnostic framework for creating parameterized design tools that use machine learned models of user intent to guide exploration through high-dimensional design spaces. The combination of a design gallery with a model of intent creates an interactive exploration interface that is more closely aligned with how the design process works. An implementation of the design adjectives system based on Gaussian process regression is presented. This implementation is able to rapidly learn user intent from only a few examples, and can generate samples of desireable designs for gallery viewing at interactive rates. Components of this framework can be improved on a domain-specific basis, and and example of such an improvement is examined in a theatrical lighting design context. Information gleaned from these exploratory design methods can be used in conjunction with parameter identification tools, such as the Hover Visualization tool presented in this thesis, to help support fine-tuning. In user studies evaluating these systems, users felt that they were able to explore the design space more quickly and easily when compared to existing per-parameter interface.

Acknowledgments

I've got a lot of people to thank, so I'll get right in to it. First, thanks to the advising team Kayvon, Jim, Sylvain, and Matt for guiding me through this long and winding process. Thanks to Chris, Jenny, Ravi, Nick, Arjun, Mark, Vidya, Rohan, Katherine, Ella, Ticha, and all of the CMU Graphics Lab past and present for sharing the burden of graduate school with me. Thanks to Sophie, Matt, David, and Nico for helping me with my sidequest to become a dragon. Special shout out to the old undergraduate crew, Ivy, Rebecca, Conrad, Matt, Maddy, and Katy, who I all miss quite a bit in the age of quarantine in which this thesis was written (maybe someday we'll all end up in the same place again). Thanks to the SCS administrative staff, especially Deb for helping me navigate through adding an advisor, Christina for helping me with department related travel, and Catherine Copetas for helping me out for just about a decade at this point (I am sorry for not showing up to my defense in my dragon outfit).

A huge thanks to everyone in my online social groups keeping me sane over the past six years. Thanks to Luvi, Yori, Tyo, Rena, and all of the Lunar Legacy FF XIV Free Company on Leviathan; Kris, Gretty, Iruku, Whitty, and all of Blitz past and present; Professor Bobo, Crow, Linehouse, Snag, Bumble, Winter, and the rest of the Beatbox denizens; my D&D group, Adam, Sara, Tyo, Jenny, Deriaz, and Crucible; Kay, Noel, Rosstin, and the rest of the west coast D&D crew who are responsible for getting me to start my own campaign; Adam, Peter, Sara, Tyler, and the rest of the Vidya discord; Jaye and Ashes from the knit scale mail crew; Crucible, Deriaz, Goo, and the rest of the dragon crew; and Aurick and Dominic, my roommates, for putting up with me. Thank y'all for being around and hanging out.

And of course, thanks to my parents and brother for supporting me through this process.

Art by @Mirsathia on Twitter

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Parameterized design applications are found everywhere in computer graphics. These parameters are used to customize the output of a visual design, such as changing the width or height of a font, or the color and number of bricks in a material in Figure 1.1. In these spaces, creating a design can be thought of as finding a point in the space that satisfies the a design goal defined by the user. Parameterized design spaces are used to control the appearance and movement of physical simulations [Gar19, Tec19, Epi, Sid20, Aut, Gol], scene and object appearance with lighting and textures [Fou20, ETC20, MA , Tec19, Sub19a], image effects [Kai92, Fac20], and even character customization in video games [Squ20, NCS16, BAN18, Cry10, Yuk19, EE19, Bli04, Bet15, Zen14, Ban19, Cap12, CP04, Are02, Cap18, Pho19, Max18, Max14, Bio09, Bio07, Nin13, Mas19, Fro15, Nev04]. Each of these design spaces has a unique parameterization, and modern content creation tools even allow designers to define their own parameterizations. For instance, materials created in Substance3D [Sub19a] may have completely different parameterizations, even if they look the same. Despite the differences in parameterization between each of these unique domains, solving a design problem is a matter of locating a point in the high-dimensional design space defined by the domain. This thesis is about creating interfaces that help designers find these points more productively.

On the surface, creating a design in one of these *parameterized design spaces* is simple; the parameters just need to be set to the values that produce a desireable output. However, defining "desireable" in this context is tricky because design preferences are often a combination of external constraints (e.g. from an art director, physical constraints) combined with personal preferences about design aesthetics. These human-driven constraints can change at any time, further complicating the *design problem*. Furthermore, these design spaces are often extremely complex, both in terms of the number of parameters involved (typically ranging from tens to hundreds), and the complexity of effects that are created by changing the parameters. One might argue that to solve this problem we should reduce the number of parameters used in each of these domains, but having a high-dimensional parameterization of a design allows it to be customized and re-used in more situations compared to a low-dimensional parameterization. For example, a parameterized font [Pro19] lets designers change the character width, thickness, and serif characteristics with a set of parameters, allowing a designer to re-use the same template in many different situations (Figure 4.12). If this template had fewer parameters, the range of fonts that could be created would be reduced. The ability to customize

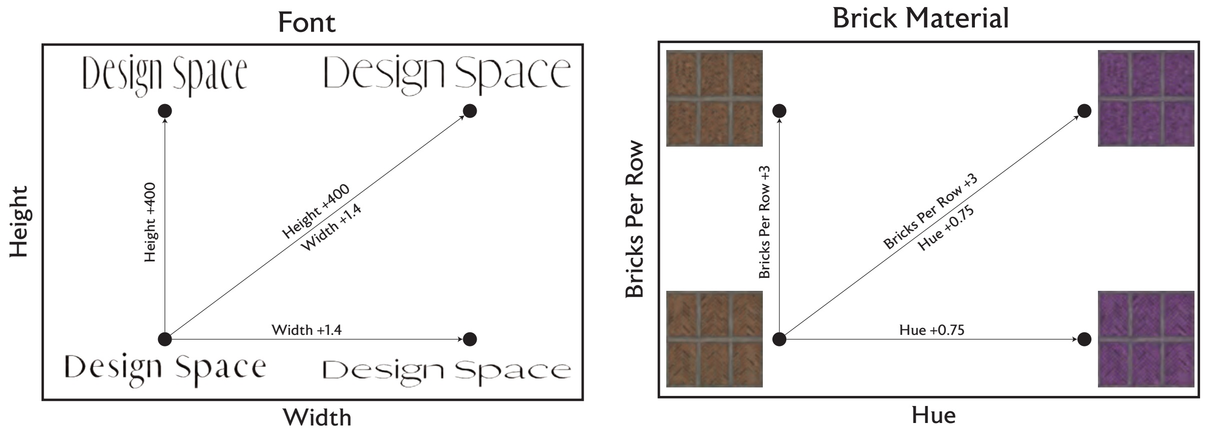


Figure 1.1: *2D Design Spaces.* Example design spaces for a parametric font (left) and parametric brick material (right). The two parameters illustrated here are a subset of a larger range of parameters for these domains.

designs to specific context is a feature highly valued by artists and designers.

1.1 The Iterative Design Process

The process of locating a design follows a common iterative *design process*. This process has three distinct phases: preliminary design, refinement, and detail design [GP92]. In the *preliminary* design phase, the designer wants to gain an understanding of what is possible in the space, and as such, this design phase is characterized by the generation of a large number of diverse design options that explore the full range of possibilities in the design space. Some of these preliminary designs are then *refined* to further develop and explore the design in context of the designer's current goals. Eventually, the designer ends up with a design that only needs a few more tweaks, and adjusts individual parameters in the final *detail* design phase. During each of these phases, the design goal itself often gets modified, as the designer better understands the requirements of the task and the capabilities of the design space. The needs of designers in each phase are different; the preliminary phase involves generating a large number of designs spanning the design space, the refinement phases takes the preliminary designs and generates variations of designs the designer likes, and detail design requires the ability to make fast, precise changes to finalize the design.

These phases themselves are composed of iterative design loops. In each iteration, the designer determines how to modify the design in order to move closer to the goal, performs the modification, and then evaluates how well the modified design met the original design goal. Sometimes, the goal itself is updated, as the designer re-considers how the current design fits their overall vision. They continue to iterate until they are satisfied. The desired actions in each of these iteration loops varies based on where the designer is in the design process. A preliminary design step often requires the ability to rapidly generate a spread of plausible designs, while a detail design step may only require the designer to identify and modify a single parameter.

As a toy example, consider a designer picking the color palette for a brick texture (illustrated in

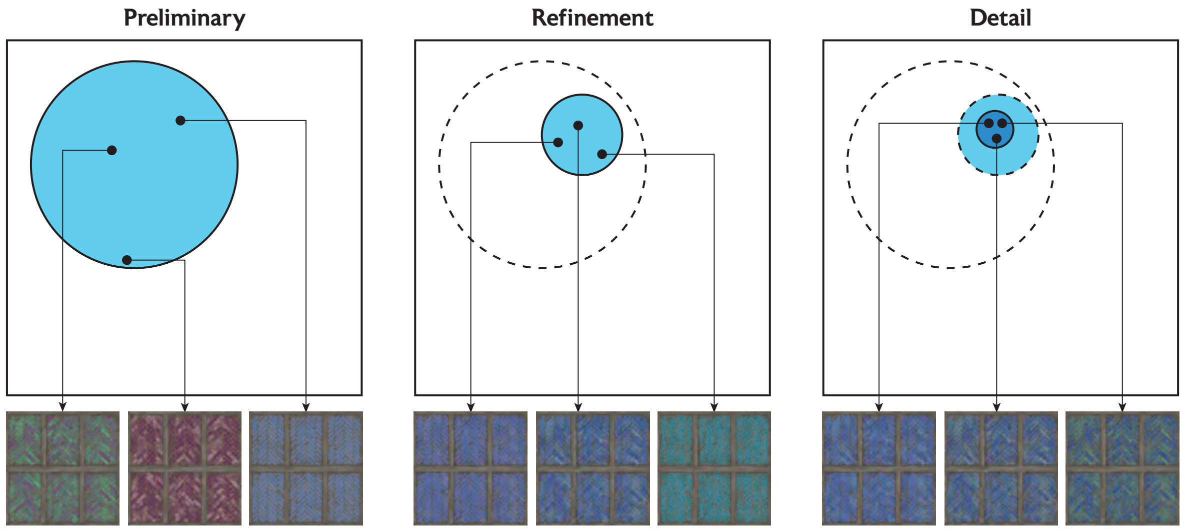


Figure 1.2: *Design Phases of a Blue Brick Material.* Conceptual illustration of the range of designs considered in each phase, and the relevant area in the design space. The preliminary phase considers broadest range of possibilities in the design space, including some designs that may not meet the design goal, the refinement design phase narrows down the range of viable designs, and the detail design phase involves small tweaks to find a final design. Note that a full design process involves multiple steps of each phase, and sometimes involves repeat phases as the designer re-evaluates their goals.

Figure 1.2). Let's say that they want to make it "blue." They will first see what options result in generally blue bricks (the preliminary design step). They then focus on a few options that have the right shade of blue and generate variations of those hues (the refinement design step). After refining the preliminary designs, the designer picks one and makes a few final per-parameter tweaks to perfect it. During this process, the designer may end up creating a design that differs from their original idea of "blue bricks." While this is a simple example showing a linear progression between the design phases, it nonetheless illustrates the process of solving a design problem. A more complex example of a professional design process is shown in Figure 4.13.

Creativity support tools aim to help designers navigate and interact with digital design spaces. Given that the design process is an iterative cycle of design creation and refinement, what should these tools do in order to aid design creation in parameterized design spaces? Looking at the requirements for each design phase provides some answers. The preliminary stage of the design process is about exploring the possibilities of the design space as quickly as possible, so speed is more important than accuracy. At this stage, tools that generate a large number of possible designs that are relevant to a rough, preliminary, design concept are particularly useful (design gallery-like systems in computer graphics are commonly used [MRR+97]). Tools that allow users to exploit a particular area of the design space that is consistent with their current design concept would be useful here. For the detail design phase, designers need to locate specific parameters quickly, so tools need to provide support for determining parameter effects.

In order to determine how well a design meets the current objectives, a designer needs to be able to see the output of the design system. This suggests that every stage of the design process should strive to render results and suggestions as quickly as possible. Similarly, methods employed for the exploratory design phases should generate and display design suggestions at interactive rates. The more designs that can be visually inspected, the better. Taken together, these observations suggest three design principles for creating interfaces for parameterized design:

Example Guided Exploration Based on Personal Intent Designers routinely create variations of designs during the preliminary and refinement design phases. Design interfaces should then be able to generate preliminary designs that span the possibilities in the design space, using learned models of intent in order to generate design variations that are of most interest to the designer. These methods should present multiple options to the designer, e.g. as a design gallery, and run at interactive rates. Users can then provide feedback to the system to direct it towards regions of interest to them, requiring the system to maintain some representation of the current design goal, e.g. through a machine learning model, or use of a model based on domain-specific design principles. While accurately modeling user preference may be difficult, in this part of the design process accuracy can be sacrificed for speed when generating designs.

Keep Access to Low-Level Parameter Controls for Fine-Tuning Once the user has been guided towards an interesting part of the design space, the per-parameter controls can be used to fine-tune the design. Here, the tool should help the user determine what each individual parameter does. Detail design can be supported by providing parameter identification tools, giving information that helps designers determine what the effect of a parameter is on the overall design. Information gained from the designer-directed sampling tools may be used to augment per-parameter controls, for example, the most impactful parameter found during the exploratory phase could be highlighted. This final detail design phase is a critical part of the process, and removing these controls would adversely affect the process for expert users, even though novice designers may not utilize per-parameter controls.

Maintain Interactivity for Fast Iteration Interactive systems are important for keeping artists in a creative flow state [Csi90], and are especially critical in computer graphics design applications, where the primary source of feedback to the designer is visual. All actions performed by the interface should show visual changes as soon as possible, ideally in under 500ms [LH14]. Rapid feedback in complex design spaces allows designers to perform more design iterations through faster design suggestion generation and real-time feedback when manipulating a slider. The faster these operations complete, the faster a user can evaluate a design and determine their next steps.

1.2 Contributions

This thesis presents the Design Adjectives framework, a domain-agnostic software framework for building parameterized design interfaces in a way that aligns with the previously outlined design principles and with the design process itself. The framework makes it easy to build such creativity support tools for multiple domains, and provides a structure to analyze how improvements should be made to domain-specific instances of these interfaces. Given the above, the thesis statement for this work is:

Combining interactive models of intent with example-baed galleries yields productivity-enhancing parameterized design tools that help users iteratively find and refine desired designs. Such tools can be scaffolded with the design adjectives framework, a domain-agnostic framework that affords domain-specific customization that can yield further benefits.

This thesis makes the following contributions.

1. **Design Adjectives: A Framework for Interactive Model-Guided Exploration of Parameterized Design Spaces** (Chapters 3 and 4) Design adjectives is a framework for creating parameterized design interfaces that use a model of intent to drive interactive exploration. The model, a design adjective, is an approximate model of a design concept that can be interactively sampled from and updated, while still retaining access to low-level parameter controls. Low-level parameter controls are augmented by data learned from the design adjective, highlighting the parameters most relevant to the current design idea. A domain-agnostic implementation of design adjectives was created and used in three design domains: fonts, particle systems, and materials.
2. **Domain-Specific Customization: Theatrical Lighting Design** (Chapter 5) As an example of a domain-specific adjustment to the general design adjectives framework, the *Visual Objectives* system allows stage lighting designers to quickly bootstrap theatrical lighting designs from a single image. The system draws from domain design principles in order to create a lighting design model that can be used with a Gibbs-sampling-like method to generate a gallery of plausible designs within seconds. These designs can then be refined and fine-tuned for use in a theatrical production using standard per-parameter control systems. This system was evaluated through a series of user studies demonstrating its effectiveness for generating plausible theatrical designs, and a case study where the interface was used for a main stage production at the Carnegie Mellon School of Drama.
3. **Hover Visualizations: Identifying Parameters with Inline Visualizations** (Chapter 6) Focusing on per-parameter control, the *Hover Visualization* system is a simple modification to image editing interfaces that directly displays the effect of layer on the composition itself. The system assists users by revealing which layers contain pixels that affect a selected region of the canvas without making permanent changes to the current layer settings. This system was evaluated through a user study, and suggests ways to augment per-parameter controls used in the two prior sampling systems. Hover visualizations exist outside of the design adjectives framework, but are still extremely helpful when performing parameterized design tasks. In the design adjectives framework, it can be used alongside relevance highlighting to help designers identify the effects of individual parameters.

Chapter 2

Background

As we saw in Chapter 1, parameterized design applications are found everywhere in computer graphics. Given their prevalence, they have unsurprisingly been the subject of decades of research in both the computer graphics and the human computer interaction communities. Before we dive into that pool of prior work, we'll need to have a baseline understanding of what the properties of a design problem are, and how designers handle the challenges of working with these ill-defined problems.

2.1 An Brief Overview of Design Problems in Computer Science

The history of the field of design in general is long and storied (and goes back at least 160 years in the manufacturing industry alone [Bay04]), but this section will take an abridged view, focusing on the necessary thinking needed to understand what a design problem is in a parameterized computer graphics context. Readers interested in more general design theory and design thinking are encouraged to follow the references in this section.

2.1.1 The Structure of a Design Problem

Computers are known to be good at solving well structured problems. A well structured problem, as defined by Herbert Simon in 1973 [Sim73], generally has a definite evaluation criteria, has a single working problem space that contains representations of all possible problem states, and is solvable with a practical amount of compute time. Simon argued that many seemingly well structured problems (such as chess) were actually *ill structured* because they did not fulfill all requirements of a well structured problem (Simon used solving chess with an exhaustive state space search as an example of an impractical amount of compute). This was particularly of interest to the emerging study of artificial intelligence, as Simon and other considered how computer systems should handle these types of underconstrained problems.

The field of design encompasses a large number of disciplines, and this thesis is primarily about purely creative parameterized design, where the final design is driven more by subjective designer preference than physical constraints. Subjective preferences are difficult to model, as they vary from person to person, and are often dependent on the particular design context being worked in. The evaluation criteria in this case is indefinite, making parameterized design an ill-structured problem. While the problem space is well-defined (every design is representable by a vector), the objective function depends on the opinion of one or more human designers. Successfully creating a result that satisfies such an objective is the process of *design*.

2.1.2 Design Thinking

While defining a design problem is relatively simple, understanding how one solves such a problem has created an entire field of research, often referred to as *design thinking*. Design thinking describes the processes used by humans to solve design problems, derived from observations, interviews, case studies, etc. of expert designers over the course of decades of human-centered research. The description of the design process has some variations across different domains [Cro11, Dor06, Sim08, GP92, Bux10], but all descriptions agree that the process of design is an iterative cycle of generating and evaluating different designs against design goals that get continuously updated as the process proceeds. In a sense, solving a design problem requires a designer to figure out both what the true design constraints are, and find a design that satisfies those constraints at the same time.

Of the cited descriptions of the design process, I have found Goel and Pirolli's description of the process [GP92] to be the most useful when talking about parameterized design. They find that the design process can be categorized into three distinct phases: preliminary, refinement, and detail design. Preliminary design involves the creation of a wide range of possible designs that satisfy an initial design goal. During this stage, the design goal itself is often updated, in a process Goel and Pirolli call *problem structuring*. Refinement involves the selection of a preliminary design, and a series of further adjustments in order to bring the design closer to a final state. Problem structuring can occur in the refinement stage, but usually to a lesser extent than in the preliminary phase. Detail design involves very low-level tweaks, as the design goal becomes fixed and the design is finalized.

To determine what interface features would help users accomplish their tasks, we can look at each phase from an interaction design viewpoint with Norman's Seven Stages model of interaction [NN13]. Norman's model maintains that a user interacts with the world, the design artifact in a parameterized design context, through a series of execution steps (the "Gulf of Execution") that attempt to achieve a goal, and a series of evaluation steps (the "Gulf of Evaluation") that determines how effectively the world was changed relative to the original goal. The model suggests that making it easier to make effective changes that move the world state closer to the design goal or making it easier to determine if the change was effective should improve the experience of using a design tool.

Viewing the three design phases through this lens leads to a number of possible ways to create better parameterized interfaces. The prevalence of generating and testing preliminary designs suggests that an interface that assists the user by automatically performing this generation according to some sort of design idea would be useful. Likewise, it would logically be useful to be able to direct that sampling as the design gets refined and the design idea gets more specific. The former also suggests that the generated suggestions do not need to be 100% accurate, as they will naturally be refined and tweaked during the design process. The constant presence of the detail design phase implies that per-parameter controls should always be accessible, as they provide the most fine-grained control over the design. Evaluation of designs in a graphical application is done visually, so any parameterized interface should strive to render results as quickly as possible. Many of these principles are echoed by a NSF workshop report on digital creativity support tools [Shn02], which advocated for exploratory design tools that are accessible to all skill levels and support different workflows. It turns out that implementing systems that follow these principles has been shown to lead to effective design interfaces, as demonstrated by the three prototype systems in this thesis.

There are, of course, many different views of the design process, seeking to understand and describe in more detail how these processes work. Dorst describes the problem structuring process in terms of resolving paradoxes [Dor06], Darke describes the iterative cycle in terms of a structure called the "primary generator" [Dar79], Cross explores the process in terms of systems, frames, and first principles [Cro11], and Simon describes design as solving a series of smaller well structured problems [Sim73]. These are all useful structure for investigating the cognitive processes driving the design process; this thesis takes these findings as a given, and leaves the details of how the design process works on a psychological level to the experts in the design thinking field.

2.2 Design Tools in Computer Graphics

The only thing necessary in an interface for parameterized design is a method for changing the parameter values. With a small number of parameters (less than 10), this kind of *per-parameter* interface is sufficient, but they tend to struggle with the curse of dimensionality [Bel57] as the number of parameters increases. The goal of many systems in computer graphics trying to address this *parameter setting* problem is to set as many parameters as possible with as few interactions as possible, thereby removing the tedium of exploring a high-dimensional space by modifying one parameter at a time. Achieving this goal in computer graphics has been the subject of 30 years of research.

2.2.1 Galleries and Sampling

An interface called Design Galleries [MRR$^+$97] is one of the most prominent exploratory design systems in computer graphics. A design gallery system samples a parameterized design space based on a sampling method provided by a user. These samples are then arranged based on another method provided by the gallery designer and displayed as thumbnails. Arrangement methods are typically 2D projections or hierarchical grouping of the resulting samples. Often, the arrangement method and sampling methods are difficult to define a priori. Gallery interfaces provide a way for users to quickly survey the possibilities in the design space, and are thus primarily used during the preliminary design phase.

Galleries need to be populated by sampling from a design space, and because of this, they are susceptible to the curse of dimensionality, as exhaustively sampling the space becomes infeasible as the number of parameters increases. Additionally, even if the space could be adequately sampled, visually sorting through thousands of returned results would be difficult for a user. Subsequent improvements to design galleries have thus focused on more efficient sampling methods, attempting to locate regions of interest in the space with human-provided seed data [RKK11, LSK$^+$10, TGY$^+$09], developing models that can be sampled from efficiently [FRS$^+$12, LRFH13], and using crowdsourced data to direct sampling towards likely regions of interest [MGB$^+$18, CKGF13]. These solutions are useful primarily for novice users, and will naturally guide users towards the consensus idea of what "good" means in a design space.

The visual nature of gallery interfaces makes them well-suited to computer graphics applications, however, they provide limited support for refining designs after the initial gallery is generated. This is not a flaw with the presentation of the results of the gallery, but rather the sampling methods used to populate them. This thesis proposes two methods that present results in a standard gallery interface, but use sampling methods that can be directed by the user. The design adjectives framework (Chapter 3) directs sampling by re-learning the current objective function every time additional feedback is given, while visual objectives (Chapter 5) use an image to define a model that identifies a region of the space that can be sampled from. Both of these sampling methods run at interactive rates, and return a gallery of results for visual exploration. This approach is conceptually similar to interactive optimization, which will be discussed in a later section.

2.2.2 Dimensionality Reduction and Semantic Attributes

One approach to making it easier to set a large number of parameter is to just have fewer parameters. Each design in a parameterized design space is a point in the high-dimensional space where each parameter defines an axis in the space. The most natural thing to do is to run a purely mathematical dimensionality reduction algorithm such as principal component analysis [F.R01], non-negative matrix factorization [LS71], linear discriminant analysis [CWA14], or t-distributed stochastic neighbor embedding (t-SNE) [MH08] using a standard distance metric (such as the L^2 distance). While these methods do reduce the number of parameters in the space, the resulting axes are often not designer-understandable, as standard difference metrics may not properly describe the differences between designs. The resulting reduction may also inadvertently cut off parts of the design space, where the designs cannot be expressed as a combination of the reduced set of parameters.

Developing better difference metrics for dimensionality reduction in parameterized design spaces is often done by gathering data that has been labeled by a human [MPBM03]. This process results in the creation of semantic attributes, which provide human-understandable axes along which a design space can be explored. Semantic attributes are a concept drawn from computer vision [FZ08, FEHF09], where a machine learning system is able to recognize both a type of object and what properties the object has. In computer graphics, semantic attributes have been transformed from a way to retrieve items from a database into a method for exploring and generating designs in a parameterized design space. Semantic attributes support the design process by providing a smaller set of parameters to work with, which does help with exploratory design by changing multiple parameters at once. These types of systems sometimes impede the detail design phase, especially if there is no mapping from semantic attributes back to the fundamental design space parameters.

Creating a semantic attribute space follows a standard set of steps; get a representative sample of the full design space, collect labeling information from humans, and train a model to convert a parameter vector to an attribute strength vector. The dimensions of the learned model then correspond to the presence of one of the learned semantic attributes, allowing users to, for instance, drag a slider to make a cloth look more "stiff" [SMD$^+$15]. This approach has been used with success for re-parameterizing domains such as shape editing [YCHK15], 3D avatars [SQRH$^+$16], image editing of outdoor scenes [LRT$^+$14], 3D component modeling [CKGF13], cloth simulation [SMD$^+$15], fonts [OLAH14], robot movement patterns [DAM$^+$19], and material appearance (BRDFs) [SGM$^+$16].

Semantic attributes provide designers with the ability to quickly survey the design space, at least within the domain of the attributes that the system understands, yet these methods come with a set of problems of their own. Gathering the training set for semantic attribute methods can be difficult in extremely high dimensional spaces due to the volume of data required, as the system will only be able to accurately estimate attribute values based on the samples that it sees in the training set. Furthermore, due to the data collection requirements, it is difficult for designers to add new attributes or customize the learned attributes (although attributes can be mixed together after training [PG11, KPG15]). In some of these systems, it is unclear how much benefit the re-parameterization provides over a simple database containing the samples used as training data combined with a content-based retrieval system [LSDJ06].

An important assumption underlying all semantic attribute methods is that the "interests of designers in each domain are regular" [BBdF10], that is, there is an assumption that there is a consistent understanding between designers regarding the meaning of an attribute. This assumption is often true in commonly studied domains, such as photo adjustment [KSSI17], but is not true in primarily creative domains, such as lighting design where designers could not agree upon the definition of attributes such as "romantic" (Appendix B.1.2). Using methods that adapt attributes to user preferences in these domains is then not possible, as no attributes exist to be adjusted. If attributes are to be used in these domains, interfaces must provide a way to quickly scaffold an attribute based on individual designer preferences, possibly without the volume of data expected by semantic attributes systems.

As we learned in Section 2.1.2, the design process is a cycle of generating and refining designs while also adjusting the design goal itself. If the design goal changes too much, semantic attributes (and combinations of attributes) may not be able to create designs consistent with these changes. If we accept that each designer or design team has their own set of preferences while creating designs, it suggests that there is no single set of attributes that could satisfy every designer. Instead, there is a need to build tools that are able to refine existing definitions of attributes or define new attributes as they work in parameterized design spaces. One such system is presented in Chapter 3 of this thesis, which has the added benefit of being able to function in spaces that are custom-created or do not have much existing labeled data, such as materials created in Substance3D [Sub19a].

2.2.3 Optimization

In contrast to gallery-based methods, optimization methods assume that a design preference can be encoded in a mathematical objective function, which can then be used to find a set of optimal designs. These systems require the definition of an objective function, which may or may not match what the designer's true objective function is. Designers might perceive differences in objectives as the system taking away control from them. In order for these systems to work successfully, they need to be able to accurately represent the range of possible design goals for the task they are designed for. Some approaches take a direct approach to modeling design constraints with an explicit objective function [YYT$^+$11, KPC93, SW14], others take user input and try to map aspects of the input to a target design [HJO$^+$01, KP09, HLCO10], and some create models of the properties of the designs in the design space [LRFH13, SSCO09]. These methods find the most success when modeling physical constraints, as in [YYT$^+$11], or use gallery interfaces to visualize multiple solutions to the optimization problem [LRFH13].

If we had a perfect representation of a design objective function, optimization methods might work rather well, although there is no guarantee that such a function could be optimized successfully. We do not have such a representation for creative tasks, so these methods will often spend a lot of time finding the best solution of an imperfect representation. This leads to a sense of user frustration if the one result output by an optimization method does not exactly match what the designer wants ("If it takes so long to run, surely it knows exactly what I want?"). Long optimization times slow down the iteration process, as users must wait for the operation to complete before being able to evaluate whether or not the process was successful. This can be mitigated by presenting multiple options in a gallery (Section 2.2.1), displaying incremental steps of the process in order to allow user modifications to the objective function while its running, and making sure the original parameters of the design space are accessible to a user.

2.2.4 Interactive Optimization

In contrast to global optimization approaches or models, interactive exploration methods rely on a feedback loop between the user and the system. One of the earliest interactive design systems was proposed by Kochar [Koc90] in 1990, prior to the publication of Design Galleries. In this system, Kochar describes a computer-aided design system which uses a model of design principles in order to suggest next steps to a user. Similarly, Hepting presents a generalized system for interactive exploration of an arbitrary parameterized design space [Hep03, HGMS04, HG05]. These works are a little obscure in the graphics literature, and were perhaps overlooked due to the intense computational requirements needed for real-time feedback or rendering of the graphics design task at the time.

These systems are conceptually similar to methods used for content retrieval and search called relevance feedback [Sal71], which takes user preferences learned during the interaction with the system to retrieve better results in subsequent searches. These methods have shown promise in content-based image retrieval systems [FSdST$^+$11]. One could envision a version of this algorithm that retrieved designs, instead of images. In data visualization, some mixed-initiative interfaces (part manual, part automatic gallery creation) systems have been proposed for exploring complex data sets, with positive results [WQM$^+$17].

Interactive optimization is an appealing concept for creative design tasks, as the process closely matches the design process itself. An interactive system would let the user take on the role of the objective function, and allow them to direct the system towards increasingly interesting parts of the design space. An interactive system does not need to have a pre-defined objective function (although it may be helpful to have rules that focus sampling towards valid designs), and instead it works with the user to define a set of criteria. Now that computers are orders of magnitude more powerful than they were when Kochar proposed their prototype, the concept of an interactive design optimization system is revisited in the design adjectives system in Chapter 3.

2.2.5 Guided Exploration

Guided exploration refers to a type of interface that presents the user with visual indications of problems with the current design or suggestions of where to take a design next. This feedback allows the user to limit their design exploration to viable regions of the space, and are most commonly used in 3D modeling domains where there are a set of physically valid constraints that must be met [UIM12, MTN^{+}15, BHU^{+}19, SCGT15]. Guided exploration applications in parameterized domains have commonly occurred in image editing, utilizing crowdsourced data to highlight regions of the parameter space that are likely to result in good adjustments [KSI14], predict what type of enhancement a user is likely to make [KCLK14], and re-parameterize portrait lighting into a small set of "basis lights" to assist with relighting a photo [BPB13]. Most similarly to design adjectives, sequential galleries [KSG20] allows users to explore designs through a series of design galleries that adapt to their detected preferences over time.

Guided exploration systems are useful for enforcing design principles and visually showing the user what the impact of those principles are. These systems are a middle ground between interactive optimization and detail design interfaces. Interactive optimization systems could be viewed as a type of guided exploration system that attempts to learn what the user's design function is, instead of just providing tools to help guide the user towards valid configurations. Guided exploration systems are difficult to use with the creative parameterized spaces found in this thesis, primarily because there are no physical constraints to easily block of infeasible parts of the design space, however the techniques found in prior work may be useful for providing better per-parameter detail design support.

2.2.6 The Last Mile: Detail Design

Tweaking and fine tuning a design is a fundamental part of the design process that cannot be overlooked by any design system. I have yet to meet a designer that did not want to make small changes to a design, that are almost invisible to other people, when finishing a project. In parameterized design, the detail design phase has been studied primarily outside of computer graphics, as the field views this mostly as an interface problem.

In HCI, a few systems have been developed to support content manipulation in visual environments. These systems include Toolglass and Magic Lenses [BSP^{+}93], which present windows that preview sections of inline content to help users make decisions faster, and Side Views [TM02], which take a similar approach and allow users to preview the effects of various actions in popout windows. These methods are examples of feedforward [NN13] mechanisms to help users determine if the action they want to perform is one that will help them achieve their design goal. Other methods try to assist users with simply selecting the right component of the design to modify. Examples of this approach include Tumbler and Splatter [RRC^{+}06], content-aware transparency [IF04], multi-blending [BG04], and LayerFish [WKB^{+}16].

Another approach to parameter setting is to provide tools that help set multiple parameters at once, sometimes through simple grouping tools that set all parameters at once (as current entertainment lighting control consoles do [ETC20, MA]), or by optimizing a subset of parameters according to user input. This type of indirect control can be useful, if the input modality is more natural than tweaking individual parameters. These approaches need to be careful that the new representation of the design problem is more intuitive than simply working with the original parameters, as Kerr et al. discovered when evaluating a lighting system that placed lights according to user-painted regions [KP09, PBMF07].

Design systems in computer graphics have often removed low-level parameters from the set of controls offered by the tools, perhaps assuming that most users would not want to use the low-level parameters after an optimization process finishes. The projects presented in this thesis all retain access to the per-parameter controls that are standard in existing interface. The sampling-based tools set all of the parameter values transparently; there are no intermediate parameters mapping returned result to original design space. This allows for the use of all existing parameter identification tools, and a new one presented in Chapter 6. Design tools will ideally combine the best techniques for per-parameter manipulation, with tools for performing fast exploratory design.

Chapter 3

Design Adjectives

The design adjectives framework is for building parameterized design tools that help users rapidly and iteratively navigate complex design spaces. The framework enables user-guided exploration by combining machine-learned models of intent with a realtime sampler to populate example galleries. The generated examples can be used to provide feedback to the model in realtime, enabling rapid iteration cycles. Information learned during the exploration phase can be used to direct users towards impactful parameters for fine-tuning. The design adjectives framework can be implemented for multiple domains, and a successful domain-agnostic implementation is presented in Chapter 4, and application-specific adjustments to this implementation are presented in subsequent chapters.

3.1 Design Adjectives Overview

To understand how the design adjectives framework utilizes learned models of user intent to support preliminary design, refinement, and final detail, consider the following scenario: Jen, a material designer, seeks to create a shiny blue version of the fish scale material shown at the top of Figure 3.1. In this walkthrough, Jen will be following a similar design process as described in Chapter 1.

Preliminary Design The first thing Jen wants to do in the preliminary design phase is to see what material variations are possible in the 22-parameter design space. This requires the generation of a large number of diverse materials. In the design adjectives framework, Jen can use a bootstrapper to generate a set of random fish scale designs. She can then browse through a gallery of these designs and select and assign high scores to designs that best represent her vision of "shiny blue scales" and low scores to those that do not (Figure 3.1-top left). The examples provided to the adjective define Jen's initial model of intent.

Refinement After surveying the design space and establishing an initial idea for what "shiny blue scales" means to her, Jen creates a number of variations of her initial designs, refining her design ideas as she does so. In the design adjectives framework, Jen accomplishes this step by sampling from the "shiny blue scales" adjective. Jen decides to look at designs that have similar scores to her current design. (Figure 3.1-top right). Jen sees a returned sample that isn't as shiny as the others,

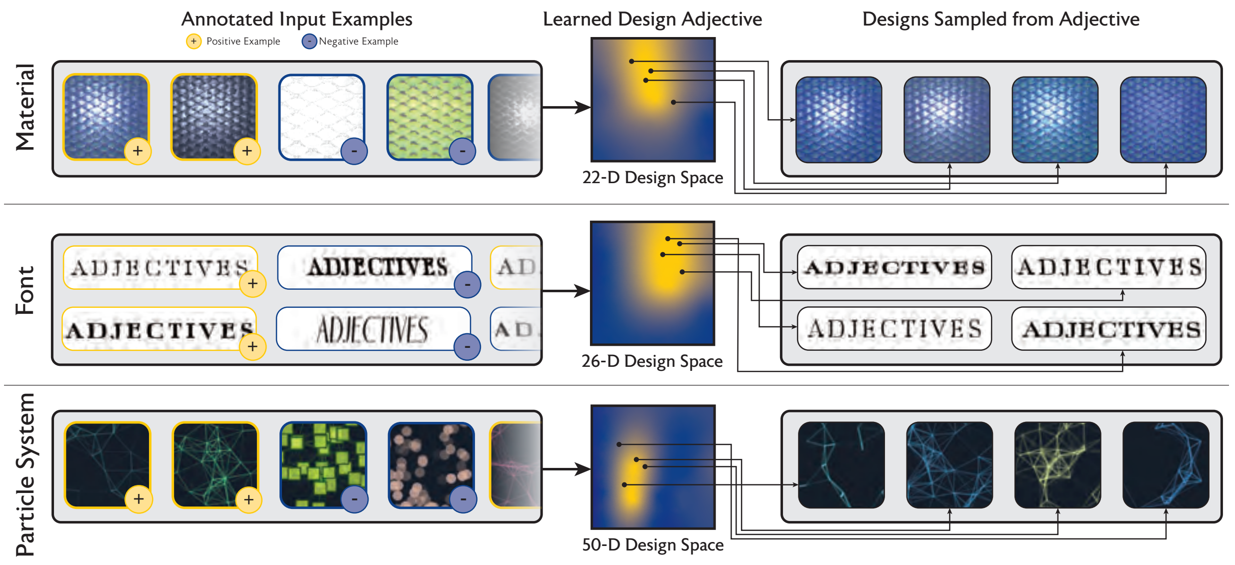


Figure 3.1: *Design Adjectives Overview.* A design adjective is a learned model of a design concept, and can be used in the design adjectives framework to guide exploration of parameterized design spaces. From top to bottom, design adjectives defined in three domains (parametric materials, parametric fonts, particle systems). Each row depicts the process of creating an adjective from a set of annotated examples in the design space (left), the online learning of the adjective's design preference function through Gaussian process regression (center), visualized here with a 2D slice of the function values indicating scores with color (0 (blue) 1 (yellow)), and a selection of design suggestions generated by sampling the adjective (right).

and assigns it a low score to update the adjective. She then runs another sampling operation with the updated adjective. The interactive refinement loop created by Jen's repeated sampling and updating of the design adjective enables intent-driven exploration of the design space.

Detail Design Towards the end of the design process, Jen wants to make a few final tweaks. To do so, Jen uses the individual parameter controls. The adjective that Jen created highlights which parameters had the most impact on her definition of "shiny blue scales," helping Jen determine which parameters to change for her final edits. Jen can make use of the framework's per-parameter tools at any time.

3.2 The Design Adjectives Framework

To enable workflows such as the one described above, the design adjectives framework consists of components that interface with design adjectives (Figure 3.2). The framework is designed to support guided exploration based on design adjectives learned from personal preferences, and facilitate access to low-level parameter tools. The interfaces specified by the framework support an interactive workflow, using real-time sampling to enable guided exploration, and always represents samples in the original parameter space to allow access to low-level parameters at all times.

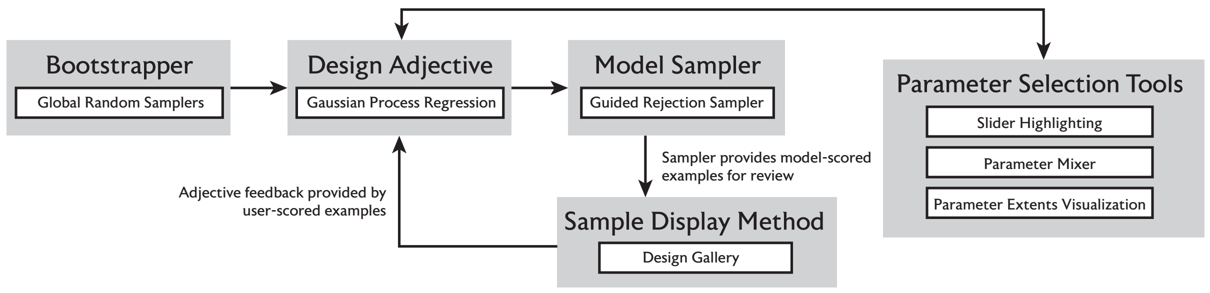

Bootstrapper
Global Random Samplers
Design Adjective
Gaussian Process Regression
Model Sampler
Guided Rejection Sampler
Parameter Selection Tools
Slider Highlighting
Parameter Mixer
Parameter Extents Visualization
Sampler provides model-scored examples for review
Adjective feedback provided by user-scored examples
Sample Display Method
Design Gallery

Figure 3.2: *Overview of the Design Adjectives Framework.* The framework consists of a set of components that allow users to explore a design space in a way that aligns with the design process itself. These components are shown in the shaded boxes, with our domain-agnostic implementation of the component shown in white boxes. The framework is centered around a design adjective, defined by scored examples, that can be used to generate new designs through a sampling method. Adjectives can be updated by adding new examples or modifying the scores of existing examples, and are initialized with the aid of a bootstrapper. Low-level parameters are always accessible. The model can be used to aid per-parameter operations, and per-parameter controls can help filter out parameters during adjective sampling.

In this section, we provide a specification for the design adjectives framework. We provide a domain-agnostic implementation of the full framework in the next section. Other implementations of the framework could modify or replace these components based on the needs of a specific design domain. For example, the sampler might be adjusted to enforce domain-specific design principles when generating designs.

3.2.1 Parameterized Design Space Definition

The design adjectives framework supports design domains consisting of a finite set of bounded, real-valued parameters. The domain also must provide a rendering function that, given a point in the design space, generates a visual preview of the corresponding design (preferably at interactive rates). The rendering function is required for enabling visual exploration, and parameter bounds are required for normalizing data passed to the design adjective. The transformation from unbounded to bounded parameters is left as an implementation detail. Our domain-agnostic implementation only requires a design domain definition for use in a new domain.

3.2.2 The Design Adjective

The design adjective enables intent-driven exploration of a design space. It models intent from user-scored input examples. The input to the adjective is assumed to be standard example-score pairings, with examples created in the design space that the user is working in. The output of the adjective is a function that assigns a real-valued preference score to all points in the design space. The adjective must be able to learn a user's intent given a small amount of initial data, and be trained and updated at interactive rates in order to support rapid iteration. The user should be able to incrementally update the adjective by providing new examples or editing existing example scores. The adjective should also provide the ability to operate on a user-specified subset of parameters within the design space.

In this context, it is important to note that the model does not need to have perfect accuracy, as mistakes can be corrected in real-time. Additionally, it should be noted that the model only needs to capture an amount of intent that allows a designer to make progress towards a design goal. Spending a large amount of time and resources to "perfectly model" a design goal is not the purpose of a design adjective.

More formally, the design adjective takes as input a set of points in the design space $\{(\mathbf{x}_1, f_1), (\mathbf{x}_2, f_2) \dots\}$, where the $\mathbf{x}_i$ points are parameter vectors in the design space $\mathcal{D}$ and the f_i values are the scores between 0 and 1 assigned by the user to these points. This data is then used to estimate the preference function $f(\mathbf{x})$ across the entire design space at interactive rates.

3.2.3 Sampling from the Design Adjective

Designs are generated from the adjective with a sampler. Generating designs serves two purposes: to show the user things that they might like, based on the current adjective definition, and to give the user the chance to assess the accuracy of the adjective and adjust the definition by manually scoring the returned suggestions.

This sampler must be able to return results at interactive rates (one second or less) according to what the user wants to see relative to the current adjective (such as "more like this" or "similarly scored designs"). Note that this interactivity requirement can be met by streaming results into the UI, allowing users to explore designs as they get generated instead of waiting for the entire process to complete. To speed up this process, the adjective must provide its training data to the sampling method, allowing the sampler to potentially use this data as part of the sampling process. For example, our implementation of this sampler uses high-scored inputs to accelerate the sampling process. Results returned by the sampler must be arranged in the UI in a way that allows users to update the definition of an adjective. In our implementation, we present the results in a gallery view.

While the specifics about how the sampler decides to return results to the gallery view are left as an implementation detail, we found four sampling methods that are sufficient for working with the sampler in our user studies. Our implementation provides a method to move *towards* desireable regions of the design space, a method to move *away* from those regions, a method to see *nearby* designs with similar scores, and a method to view a range of designs along the *axis* defined by the adjective. These sampling modes are described in more detail in Section 4.2.

3.2.4 Bootstrapping the Adjective

The adjective must use in-domain examples as inputs. This means that when a new adjective is created, there are no existing examples to use, and users must find some way to provide initial examples and scores to the adjective. At minimum, this can be done with existing per-parameter controls, but better tools can be provided. Our implementation provides a uniform random sampler as its bootstrapper. We expect that interfaces developed for a specific domain will be able to provide better bootstrappers by using domain-specific design principles.

3.2.5 Per-Parameter Identification Tools

The design adjective can be used to provide additional support during the detail design phase. The preference function can be used to enable tools such as per-parameter highlighting [KSI14] when performing detail design. All framework components operate on the original parameters, so other techniques for parameter identification [TM02, SFPF19] can be easily added to any implementation of this framework.

Chapter 4

Implementing and Evaluating Design Adjectives

We present a domain-agnostic implementation of the design adjectives framework that can be used with any design domain that meets the framework specification. The implementation has been used as the basis for creating design tools for parametric materials, fonts, and particle systems. Domain-specific implementations of this framework gain immediate access to all domain-agnostic components, and can replace these components with domain-specific variations as desired. This implementation is evaluated through a series of user studies, demonstrating the framework's capability to support expert and novice workflows.

4.1 Building Adjectives using Gaussian Process Regression

Recall that a design adjective takes as input a set of points in the design space $\{(\mathbf{x}_1, f_1), (\mathbf{x}_2, f_2) \dots\}$, where the $\mathbf{x}_i$ points are parameter vectors in the design space $\mathcal{D}$ and the f_i values are the scores between 0 and 1 assigned by the user to these points. We use Gaussian process regression (GPR) to estimate the preference function $f(\mathbf{x})$ across the entire design space at interactive rates. We chose GPR because it satisfies the requirements expected by a design adjective in the framework; it is able to estimate functions given a small amount of input data, and can be trained in real time. We perform the regression with GPyTorch [GPB$^+$18]. GPR is a known technique and we refer to the book by Rasmussen and Williams [RW06] for a detailed review. For the sake of completeness, the rest of this section introduces the basic concepts required in our context and discusses how they apply to the design adjectives framework.

GPR uses a Gaussian process (GP) to estimate the value of an unknown function. A Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution. Gaussian processes are completely specified by a mean function $m(\mathbf{x})$ and a covariance function $k(\mathbf{x}, \mathbf{x}')$. In the case of a design adjective, the random variables are the preference scores f_i of the points $\mathbf{x}_i$ in the design space. An input pair $(\mathbf{x}_i, f_i)$ can be viewed as a point sample of the preference score at $\mathbf{x}_i$. A GP defines a distribution over functions, meaning that the value of the GP at a point $\mathbf{x}^*$ in the space is itself a Gaussian distribution. We use the mean of the GP as the estimate of the adjective score at an unobserved point $\mathbf{x}^*$. We do not make use of the variance in our implementation.

To define a GP for a given design adjective, we create the matrix $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2 \dots]$ (i.e., its columns are the points with a user-assigned score) and the vector $\mathbf{f} = [f_1, f_2 \dots]^\mathsf{T}$ from the scores, and use a standard zero-mean function $m(\mathbf{x}) = 0$, indicating that every unobserved point is assumed to have a low adjective score. For the covariance function, we use the radial basis kernel:

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{x}')^\mathsf{T}\, \boldsymbol{\Theta}^{-2}\, (\mathbf{x} - \mathbf{x}')\right) \tag{4.1}$$

where $\boldsymbol{\Theta}$ is a diagonal matrix learned at training time that applies a scaling factor to each dimension of the design space. This function k is used to construct a covariance vector $\mathbf{k}(\cdot\,,\cdot)$ with the covariance values between the column vectors of a matrix and a given vector, and a covariance matrix $\mathbf{K}(\cdot\,,\cdot)$ with the covariance values between all pairs of vector columns of two matrices. Using these quantities, the GP estimates the score at a new point $\mathbf{x}^*$ as follows.

$$f(\mathbf{x}^*) = \mathbf{k}(\mathbf{X}, \mathbf{x}^*)^\mathsf{T}\, \mathbf{K}(\mathbf{X}, \mathbf{X})^{-1}\, \mathbf{f} \tag{4.2}$$

The GP defined above can actually be used as an estimator immediately, although it is not likely to be accurate, as the covariance function does not accurately represent the relationships between the different parameters. In order to improve the accuracy of the function, we optimize the GP by adjusting the scaling factors contained in the $\boldsymbol{\Theta}$ matrix of the covariance function for each parameter (Eq. 4.1), a process called *automatic relevance determination* [RW06, ch. 5.1]. Intuitively, the scaling factors indicate how quickly the function varies along each dimension. In this instance, small factors indicate that the function value varies quickly, while large factors indicate that the function varies slowly. The value of the factor associated with a parameter is a proxy for how important it is to the overall function score. After completing this regression process, the GP can be used to more accurately estimate adjective scores for arbitrary points.

Adjectives may be defined on a subset of parameters in the design space. This subset is referred to as the adjective's *affected parameters*. Using GPR to implement affected parameters is straightforward: the input parameter vectors $\mathbf{x}_i$ defined on the full design space are projected onto the affected parameter subspace before being provided to the GPR. Our implementation automatically detects affected parameters from the input, assuming that a parameter is affected if it has different values in any two input points. Users can override the set of affected parameters at any time.

4.2 Generating Designs with a Guided Rejection Sampler

In order to use the Gaussian process model of an adjective in the design adjectives framework, we need to provide a sampling method. To return samples as quickly as possible, we introduce a guided rejection sampler on the design adjective (Alg. 1), accepting samples that meet a user-specified criteria described later in this section, and are different enough from already accepted samples according to a L^2 distance threshold.

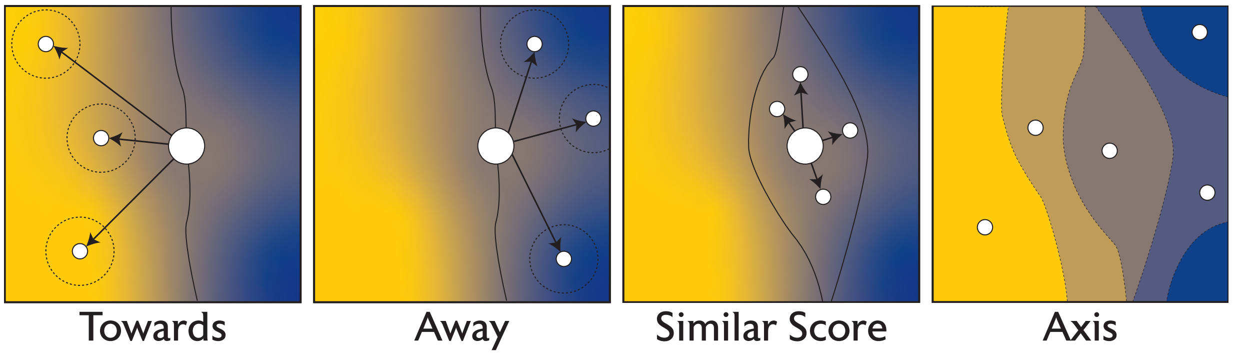


Figure 4.1: *Sampling Mode Visualizations.* The four sampling modes described in the "Acceptance Criteria" section visualized on a 2D design space. The adjective score is represented by color, with yellow indicating high scores and blue indicating low scores. The starting point for each sampling method is indicated with the large circle, and the sample points represented with smaller circles. Dotted lines around the sample points in Towards and Away represents the difference threshold criteria.

Conceptually, Algorithm 1 attempts to guide itself towards good samples by starting with highly-rated examples provided by the user (Step #1), and randomizing a subset of the parameters of those examples (Step #2). The sampler should be returning novel samples, not just recycling what the designer has already input, so we do not use a hill climbing (or similar) step in the sampler to avoid returning the best already labeled samples. In our implementation, p_0 is set equal to half of the length of the design space parameter vector, $p_{\text{floor}} = 3$, $maxIter = 10000$, $r = 10$, and $count = 20$. All five settings can be modified by the users, although none of them saw the need for it in our experiments.

4.2.1 Acceptance Criteria

Users may have different intents at different stages of their exploration. We express these intents in the sampling algorithm by changing the rejection sampler's acceptance criteria (the `Accept`$(\mathbf{x}, f(\mathbf{x}))$ function in Algorithm 1). We implement the four criteria described in Section 3.2.3 (Figure 4.1).

Towards generates samples with a higher adjective score, which is useful, for instance, when searching for the final design. New designs must score better than the current design.

Away generates samples with a lower adjective score, e.g., to refine the definition of the adjective. New designs must score worse than the current design. This sampling mode is used to get out of local maxima, moving the current design away from known good samples in order to show more variety.

Similar Score picks variants with similar "amount of the adjective" to the current design, e.g., to help users explore the design space without converging to a high-score sample. New designs must have an adjective score within $\pm 10\%$ of the current design's score. Samples returned by this method are not necessarily visually similar to the current design, as the similarity metric is the adjective score not parameter vector distance.

Axis visualizes samples regularly spaced on the adjective scale, which is useful to assess the range of the effect captured by the adjective and quickly navigate through it. All new designs are accepted

Algorithm 1: Guided Rejection Sampler

```
1  input   adjective function and input examples: $f(\cdot)$, $\mathcal{X}$
           set of affected parameters: $\mathcal{P}$
           current design: $\mathbf{x}_0$
           settings of the fine randomization: $p_0, p_{\text{floor}}, r$
           max number of iterations before bailing out: $maxIter$
           number of samples to generate: $count$
           acceptance criteria function: $Accept(\mathbf{x}, f(\mathbf{x}))$
   output  set of samples: $\mathcal{S}$
2  // Initialization
3  $\mathcal{S} \leftarrow \{\}$     $p \leftarrow p_0$     $i, p_{\text{count}} \leftarrow 0$
4  $\mathcal{X}_{\text{good}} \leftarrow \{\mathbf{x}_i \in \mathcal{X}, \text{ s.t. } f(\mathbf{x}_i) > 0.5\}$ // Select examples with a score above 50%.
5  // Main loop of the sampler
6  while $(|\mathcal{S}| < count)$ and $(i < maxIter)$ do
7      // Step #1: Coarse Randomization
           Randomly select a good example.
8      $\mathbf{x} \leftarrow GetRandomElement(\mathcal{X}_{\text{good}})$
9      // Step #2: Fine Randomization
           Randomly perturb a random parameter subset.
10     $\mathcal{I}_{\text{random}} \leftarrow SelectRandomIndices\ (\mathcal{P}, p)$ // Subset of affected indices to randomize
11     for $index \in \mathcal{I}_{\text{random}}$ do
12         $\mathbf{x}[index] \leftarrow UniformRnd(0, 1)$ // Use a uniform distribution to randomize the
             selected indices.
13     // If a parameter is not affected by the adjective, set it to the value of the current design.
14     for $index \notin \mathcal{P}$ do
15         $\mathbf{x}[index] \leftarrow \mathbf{x}_0[index]$
16     // If this is a valid sample...
17     if $Accept(\mathbf{x}, f(\mathbf{x}))$ then
18         $\mathcal{S} \leftarrow \mathcal{S} \cup \{\mathbf{x}\}$ // add it to sample set...
19         $p \leftarrow p_0$ // and reset the fine randomization.
20     else
21         $p_{\text{count}} \leftarrow p_{\text{count}} + 1$
22         // If the algorithm failed too many times to find a valid sample...
23         if $p_{\text{count}} > r$ then
24             $p_{\text{count}} \leftarrow 0$ // reset the counter...
25             $p \leftarrow \max(p - 1, p_{\text{floor}})$ // and weaken the fine randomization without going
                 below a minimum.
26     $i \leftarrow i + 1$
```

at first, but an additional de-duplication step is added where designs that score within $\pm 2.5\%$ of a previously accepted design are rejected.

4.3 Bootstrapping Design Adjectives

Adjectives are defined by labeled examples, and a newly created adjective has no examples to start with. In order to initialize, or bootstrap, the adjective, examples must be provided by the user. To assist users in this task, the design adjectives framework requires a bootstrapping tool. We provide two random samplers for generating preliminary examples in a domain-agnostic scenario.

The *uniform randomizer* draws samples from a uniform random distribution. The *jitter* sampler applies a random delta to each parameter in the current parameter vector. These functions can either operate on all parameters in the design space, or or use a subset of parameters selected by the user, as detailed in Section 4.4.2. We find that these samplers are sufficient for working in generic design spaces.

4.4 Per-Parameter Tools

As the designer reaches the end of the design process, small tweaks to the design are performed using individual parameter controls. Per-parameter tools are necessary to help direct users to controls that are most relevant to their current intent. The framework requires that the output of sampling an adjective is a design representable in the original parameter space, so all existing per-parameter tools can be used in this platform. We provide a set of tools for per-parameter manipulation that we found to be most useful for users working in general parameterized spaces. This is a non-exhaustive list of per-parameter tools, and we expect that domain-specific implementations of this platform will add additional tools specific to their domain.

4.4.1 Highlighted Parameter Sliders

We color-code the sliders to show the user how the adjective score would change if the parameter value were modified (Figure 4.2). This technique helps users navigate complex design spaces [KSI14]. To color-code the sliders, we evaluate the adjective at individual parameter values across its full range, sampled in increments of 5%, while holding all other parameters constant. Our interface supports both absolute and relative color coding of of sliders. Absolute color coding assumes the adjective function values fall within the $[0, 1]$ range and assign colors accordingly, while relative color coding assigns colors relative to the minimum and maximum values observed in the parameter sweep over all parameters. Running the color-coding operation is almost instant in our implementation, allowing users to perform optimization steps themselves, if they desire, during the fine-tuning phase.

4.4.2 Parameter Selection

Adjectives do not need to affect every parameter in the design space. Designers can select a subset of parameters to work with while defining adjectives and running the bootstrapping tools. Users may

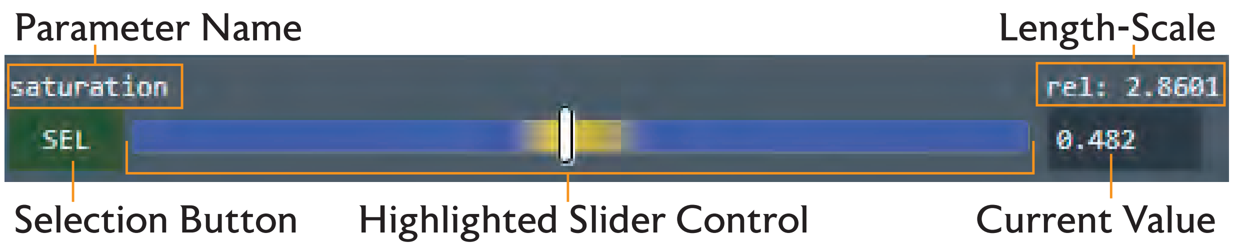


Figure 4.2: *Slider Controls.* Labeled components of a single parameter control element in our adjectives interface. The slider highlighting indicates how the current adjective score would change if the parameter were moved to the corresponding location. Yellow highlighting indicates a higher score, blue indicates a lower score.

wish to do this to reduce the dimensionality of the search space; focusing only on changing parameters that are likely to lead to a design of interest. Expert designers may not need help identifying which parameters are relevant, but novice users may have trouble determining what a parameter does.

To assist users in finding relevant parameters, we provide a visualization of the *parameter extents*. Clicking on the name of a parameter toggles this visualization, which displays thumbnails of designs that span the full range of the selected parameter's values while all other parameters are held constant (Figure 4.3). This visualization is a type of side view [TM02], providing assistance with locating the effect of a single parameter so that the user can determine if the parameter should be included or excluded in the next operation.

4.4.3 Parameter Mixer

The *mixer* utility takes two parameter vectors and creates a new vector by randomly combining parameters from the input vectors, similar to a crossover operation in a genetic algorithm (Figure 4.4). Designers can set a bias value to select parameters more frequently from one design over the other.

The mixer returns a gallery of results, each with a different set of parameters selected for combination. This allows designers to pick out elements that they like in a design, without needing to know which exact parameters are responsible for creating that element. The mixer does not require an adjective to operate. As an example, the mixer could be used to combine a blue brick design with a large brick design to create a large blue brick design, without the designer needing to know what parameters control those properties.

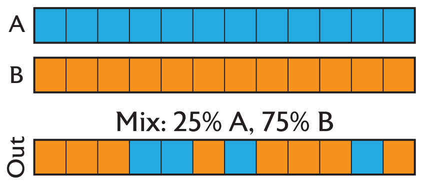


Figure 4.4: *Mixer.* One run of the mixer combining designs A and B, with a bias towards design B.

4.5 Implementation Details

We implement the design adjectives framework using a generic client-server interface. The server runs the training of the adjective model using GPyTorch [GPB+18], and the client renders designs and provides the requisite UI for interacting with the adjectives. The server does not have knowledge

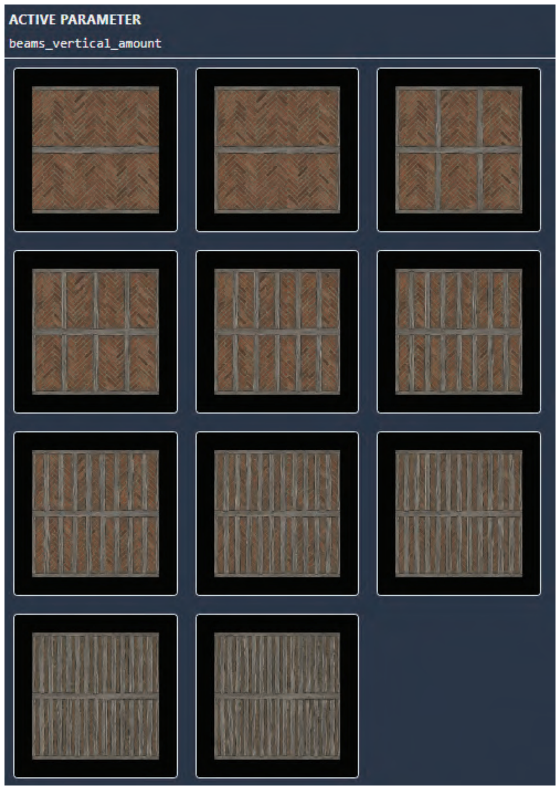


Figure 4.3: *Parameter Extents.* Displaying thumbnails for values of "beam_vertical_amount" between 0 and 1.

of any specific design domain, operating on the parameters assuming they are bounded within $[0, 1]$. The client performs the normalization step before sending data to the server, and discrete parameters are made continuous by mapping the enumerated options to equal-sized intervals within $[0, 1]$. This is a common way of performing this type of transformation, but does create visual discontinuities along the discrete parameter's range.

The server's adjective model training process favors training speed over accuracy, and the number of optimization steps taken is tuned such that the training completes in under a second (400 iterations on our hardware). Training a GPR is known to scale poorly as the size of the training set increases, typically running in $O(n^3)$ time. In our context, n is the number of input points, i.e., $n = |\mathcal{X}|$. Most adjectives defined in our implementation are well under the size at which this becomes a problem, however sparse Gaussian process regression algorithms can be used if needed [GPB+18]. Splitting the capabilities of the server and client in this way made it possible for us to implement three design domains (fonts, materials, particle systems) in the framework using the same generic software primitives.

The user interface (Figure 4.5) is written in Javascript using Electron [Ele20] to provide a desktop application wrapper. New design domains can be added to the interface by implementing a driver interface, requiring the domain to provide a rendering function for full-quality renders and thumbnail previews, and requiring a list of parameters that can be modified in the domain. Rendering is typically performed on an HTML canvas element. Implementing the three domains in the interface took a few hours of development time, with the variance coming primarily from how easily results could be rendered in the Electron-based UI. With enough development time, the server could be re-used in plugins developed for existing parameterized design interfaces. Client and server source code for our implementation can be found at `https://github.com/ebshimizu/DesignAdjectives`.

4.5.1 API Implementation Details

This implementation of design adjectives is motivated by the need to create a domain-agnostic framework for parameterized design. At the same time, we must be able to handle a large computational load, both in terms of rendering visual outputs for the domain and handling on-demand training of a design adjective. In order to accomplish this, we used the following principles when implementing the framework:

Isolate Domain-Specific Data Any domain-specific data needs to be isolated within its own component. This component is treated as a black box by the client; it provides a list of parameters and valid ranges for such parameters and handles rendering of arbitrary arrangements of its parameters. Isolating domain-specific data in this way allows the entire interface to operate in the same way for each domain, and reduces the time needed to implement new domains in the interface. We expect that domain-specific implementation may loosen this requirement and allow for domain-specific operations in both the client and server.

Distribute Workload Training a model is not a light computational load, and neither is rendering a large number of thumbnails for visual design tasks. In order to help keep the system running at interactive rates, we provide the option of splitting model training and design rendering between the server and client respectively. This client-server model also allows us to make use of cloud computing to scale the amount of compute needed for training adjectives as-needed.

For those interested in implementing a new domain or other component in this interface, additional details about the specific software interfaces used in the system will be described below.

Design Domain

Each domain is implemented in a `Backend` component (Listing 4.1) which, at minimum, provides the client with a list of parameters (including minimum and maximum values), exposes a `render` function that can render output to either a HTML `canvas` or a `<div>` element containing text, and a `loadNew` function that is given a file to initialize the backend with. Examples of implemented backends can be found in the `client/node_ui/src/renderer/store/backend` folder of the code.

```
Interface Backend {
  async void render(params, canvas, textDiv, settings)
  Param[] getParams()
  void loadNew(inputFile)
}
```

Listing 4.1: Backend component interface

The `render` function is given references to a `canvas` or `<div>` element containing text. This function is an asynchronous function that can use whatever method it chooses to modify the visual appearance of the elements. This function is called in two contexts: to render the current design, presumably using a higher-quality setting, and to render a thumbnail preview, which presumably uses a lower-quality setting. This information is provided in a `settings` object provided to the function.

The returned list of parameters from `getParams` is an `Object` array. Each object represents a parameter, and contains the following fields:

```
Interface Param {
  name : string,
  value : Number,
  min : Number,
  max : Number,
  links : Number[]
}
```

Listing 4.2: Parameter interface

The `value` field is used as a default value for when the backend is first instantiated. The parameter is normalized by the client with the `min` and `max` values. This marshalling operation is handled automatically by the client. The `links` array contains the array indices of other parameters that the parameter is considered to be linked to. This is used by the client to group parameters that should be modified by samplers at the same time. For example, a color parameter with three channels should link those three parameters together so that all channels are properly adjusted when sampling.

Adjectives

On the client, adjectives (Listing 4.3) are defined simply by a list of scored examples. This is an array of `Objects` containing two required fields: `x`, a parameter vector representing a design, and `y`, a score assigned to the design by a user. Each input point can optionally contain an `affected` field which indicates which parameters should be considered relevant for training and sampling.

```
Interface Adjective {
  name : string,
  data : Object[],      // Contains: x (float[]), y (float), affected (opt., int[])
  trainData : Object,   // Returned by server when training is complete
  trained: boolean,     // Set to true when client can call scoring functions
  filter: integer[]     // parameter indices that should be included for training
}
```

Listing 4.3: Adjective interface

Adjectives are trained on-demand. When the input list of scored examples is modified, the client sends the adjective's input points to the server. The server runs the training operation, and returns the trained model parameters. When the client sends the input points, it first filters the parameter vectors to only include affected parameters (using a union of the data's `affected` and the adjective's `filter` field). Affected parameters are either user-specified, using the per-parameter selection tools in the interface, or automatically determined. A parameter is considered to be unaffected if its value is the same across all input examples, otherwise it is considered to be affected. This filtering operation helps reduce the dimensionality of resulting models, and provides a mechanism for users to further control the exploration process.

We run the adjective training on a desktop machine with an Intel Cor i7-6700K CPU 4Ghz. The models we train on are small, so we use GPyTorch on the CPU for training. As models increase in size and complexity, the server can be transferred to more powerful machines without affecting the client. The server does not permanently store adjective information, so adjective training instances could even be created on-demand in production environments.

Trained adjectives are cached by the server. In order to sample from an adjective, it must have been previously trained or loaded into the server's cache. Adjectives store the parameters for the trained model, so if an adjective has not been modified, it can load the parameters into the server without re-training. Communication with the server is facilitated by socket.io.

Samplers and Bootstrappers

Adjective samplers and Bootstrappers use similar interfaces in our implementation. Both constructs are samplers, with adjective samplers taking the current adjective as an additional parameter. All samplers are given a parameter vector representing the current design, and are allowed to take any number of extra parameters to configure the search process. Samplers are expected to output results as soon as they are available, allowing users to immediately inspect new designs without waiting for a possibly lengthy sampling process to complete. Samples are returned through an event-based model, with the server emitting an event containing the complete parameter vector for the new sample.

Samplers are typically invoked through menu commands, which automatically select a sampling method and set the parameters for the sampler accordingly. This is how the sampling criteria in Section 4.2 are implemented. The parameters for the samplers are displayed in our interface, mostly for debugging purposes, but users are able to create their own criteria if they so choose.

4.5.2 Client and Server Responsibilities

Any function involving the computation of an adjective value, the training of an adjective, or sampling an adjective is handled by the server. All other functions are handled by the client. Rendering is currently handled by the client machine, though it could be offloaded to a remote rendering server if desired.

When an adjective is changed on the client (adding, removing, or modifying input points), the client sends a request request to the server to re-train the adjective. Once the adjective is trained, the client can then request a score of any design from the server. Sampling an adjective can only be done when the adjective has been trained, indicated by the `trained` field of the adjective (Listing 4.3).

The server does not persist any adjective data between sessions. The client can send previously used training data, stored in the `trainData` field of the adjective, to quickly prepare an adjective for training. This capability is provided by GPyTorch.

4.5.3 Applications

`Backend`s were implemented for three design domains: particle systems, fonts, and materials. These domains were all implemented in the same design adjectives user interface; the only difference is the implementation of the `Backend`.

Particle Systems

This implementation uses a version of particles.js [Gar19] with a few adjustments made to allow for easier instantiation of multiple systems at once. Rendering this domain was easy, as the particle system library uses `<canvas>` elements natively. The library uses 50 parameter values, with minimum and maximum values selected based on reasonable visual appearance (e.g. speed) and hardware rendering capabilities (e.g. number of particles).

Fonts

This implementation used a javascript version of the Prototypo [Pro19] parametric font API. Rendering this domain was a little tricky, as the library wanted to operate on text by using CSS classes instead of a `<canvas>` element. An option was provided in the implementation to render to a sample font string contained in a `<div>` or a `<canvas>` depending on the requirements of the domain. This domain used 26 parameters, with each template containing the same parameters. Minimum and maximum values were selected based on the limits displayed in the online Prototypo editor.

Unfortunately at the time of publication of this thesis, Prototypo was shut down and can no longer be used in its original form. A subset of Prototypo templates were archived by the author in order to maintain demo capability of this domain. A modern version of this domain could choose to use variable fonts instead of proprietary templates.

Materials

This implementation used Substance3D materials. There is no public browser-based implementation of the Substance engine, so instead materials are written to disk by using the Substance Automation Toolkit, and then loaded and rendered on basic 3D shapes with the physically-based shader in the three.js [Mr.19] library. This approach has a number of limitations. Rendering speed is limited by how quickly materials can be generated, written to disk, and then re-loaded in memory. The appearance of the material is not a one-to-one match with substance, as the three.js shader is incapable of exactly replicating the shader models used by Substance3D. These drawbacks could be addressed by using a browser-native implementation of the Substance3D rendering engine. The number of parameters used in this domain varied based on the definition of the material, typically ranging from tens to hundreds.

4.6 Evaluation

To determine how effectively our implementation of the design adjectives framework supports exploratory design, we perform two studies: a user study with intermediate users evaluating the tool in a series of short design tasks, and a case study evaluating the tool's ability to support expert workflows.

In the user study, we are primarily interested in determining the extent to which the design adjectives are perceived to accurately represent a design concept. We also investigate the extent to which the system is perceived by users to support exploration, and which parts of the toolkit are viewed as the most useful. We detail the process and results in the "User Study" section and find that our

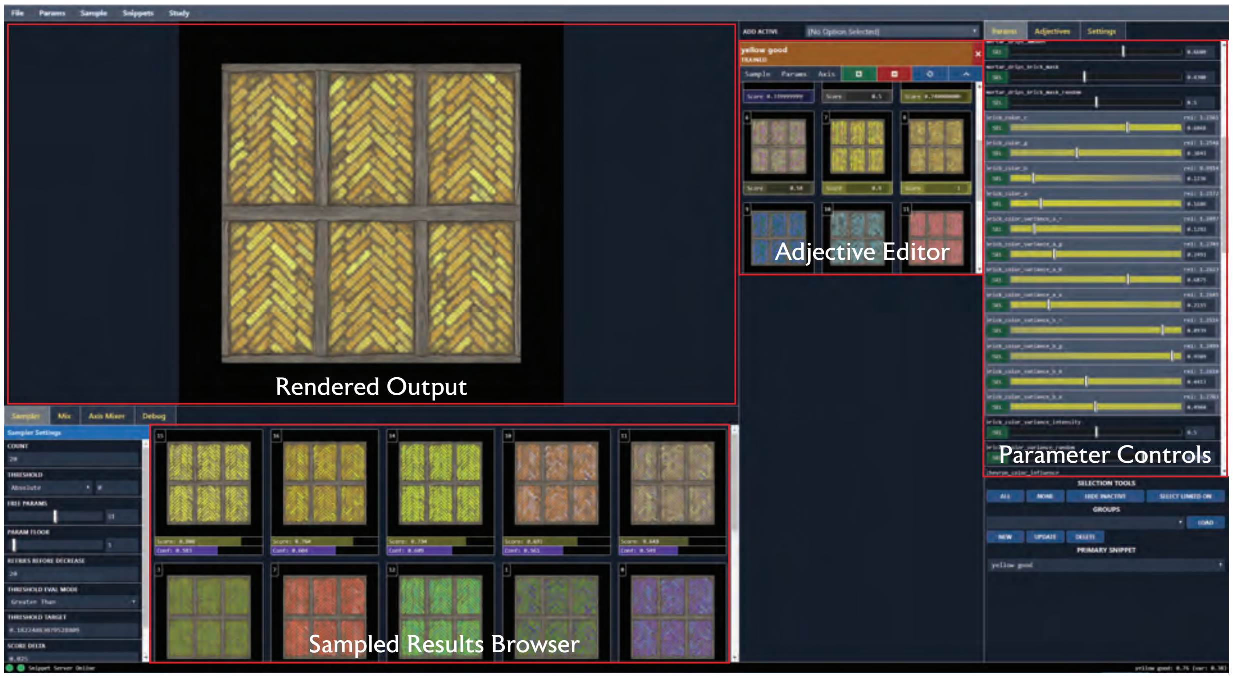


Figure 4.5: *Design Adjectives User Interface.* A screenshot of the design adjectives interface used in the evaluation of the system. The current design is rendered in the top-left. Design suggestions returned from sampling an adjective are shown in a gallery at the bottom, and hovering over a thumbnail renders the design in the main view. The adjective definition is viewable and editable in the right-center, with per-parameter controls always available on the right.

implementation is perceived as being able to represent user's design concepts, and support users' exploratory design process better than existing interfaces.

In the expert case study, we investigated the extent to which the design adjectives framework can support an expert-level design process. The tool was given to a professional graphic designer, who used it to create a series of fonts. The designer found that adjectives enabled them to perform design concepting easily and quickly. We detail the findings of this study in Section 4.6.2. In an additional informal evaluation, an expert material designer spent a day with the tool to generate a set of brick material variations. The results of this design session are detailed in Section 4.6.3.

4.6.1 User Study

In the user study, participants performed three design tasks: two using our implementation of the design adjectives framework, and one using a baseline sliders configuration. We collected user feedback via Likert scale questions measuring the extent to which users felt that each configuration helped them with the design task. Overall, we found that participants felt that they were able to use adjectives to easily explore the design space, and find solutions that they felt satisfied the design task. Participants preferred to use the adjectives configuration over the baseline sliders-only configuration for exploratory tasks.

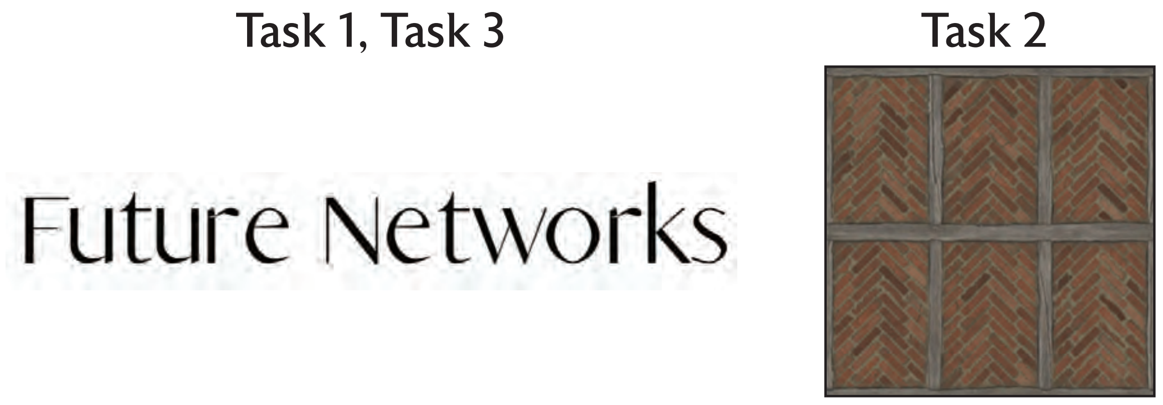


Figure 4.6: *User Study Initial Configurations.* The initial designs displayed to users performing the tasks in the user study. The font for Tasks 1 and 3 is shown on the left, and the material for Task 2 is shown on the right. The particle system configuration is omitted here because it is not used in an evaluated task, but is shown in the video accompanying this paper.

Participants

10 adult participants were recruited for this voluntary study. No compensation was given. All participants had at least used a computer graphics design tool (e.g. Autodesk Maya, Unity3D, Adobe Photoshop, etc.) before. Participants generally rated themselves as skilled users of design tools, but not daily users of these tools, with a median self-reported skill of 3 out of 5 on a scale ranging from "uses design tools once every few months" (1) to "uses design tools daily" (5). Users were not specifically knowledgeable about particle system design, material design, or font design.

Interface Configurations

Participants used the two following configurations.

Design Adjectives This configuration allows users to use our implementation of the design adjectives framework (Figure 4.5). When performing a task with this configuration, users were limited to using one self-defined adjective, mirroring the workflow described in the "Design Adjectives Overview" section. No pre-existing adjectives were provided.

Sliders This configuration only allows users to use the sliders, a baseline similar to existing state-of-the-art computer graphics tool interfaces. This configuration used the same interface shown in Figure 4.5 with all design adjectives-related functionality disabled.

Tasks

We used one design domain to teach users how to use the design adjectives configuration, and two design domains to evaluate the design adjectives configuration. Each participant in this study performed the same tasks in the same order, performing two tasks with the design adjectives configuration and then repeating the first task using the sliders configuration. By running tasks in this order, we can investigate to whether or not the design adjectives configuration helped users remember features of the parameter space, and locate interesting regions more quickly upon repetition.

Participants were first given a 20 minute tutorial, where they were taught how to use the tools present in the design adjectives configuration using a particle system. They were then given a series of three 10-minute design tasks where they were instructed to create a font or material according to a description of the design goal. Participants were allowed to finish their design tweaks at the end of the 10 minute time period, if they felt that it was necessary.

Participants answered a series of Likert-scale survey questions between each task, and the interface recorded a log of actions taken during each task. At the end of the study, participants answered three forced-choice questions asking them to choose one of the two configurations for use in general exploration tasks, directed exploration tasks, and detail design tasks. They were also allowed to give written feedback at the end of the study. The study took approximately 60 minutes to complete.

Particle System (50 parameters): Tutorial. A particle system simulation run by particles.js [Gar19], an open-source library. This domain is only used for the tutorial task. The tutorial consisted of a fully user-driven exploration task, where participants were instructed to find a particle system that was interesting to them using the adjectives configuration while being taught about the interface capabilities.

Font (26 parameters): Tasks 1 and 3. Participants were asked to create a "futuristic" font for a made-up networking conference titled "Future Networks." Participants designed their font using the characters contained in the conference title. The parameters were provided by the Prototypo [Pro19] font design system, using the "Elzevir" template as the base (Figure 4.6-left). Task 1 allowed users to use the design adjectives configuration, while Task 3 repeated the design task with the sliders configuration. Under these conditions, we expected that the repeat task would be easier, as users had previously worked in the design space.

Material (62 parameters): Task 2. Participants were asked to make the selected brick texture look more "weathered, cracked, old, or broken." The material was created using Substance3D [Sub19a] and rendered using the physically-based shader in the three.js library [Mr.19] (Figure 4.6-right). This configuration was used in Task 2.

User Study Results

We draw four main conclusions from analyzing the results of the user study:

- Adjectives support exploratory design,
- Users prefer to use the adjectives configuration for exploratory design tasks,
- Adjectives are the most commonly used component of the adjectives configuration, and
- Adjectives are used for directed exploration in particular.

Adjectives Support Exploratory Design Participants felt that the design adjectives framework helped them with exploratory design, in that it was easy to explore the different design options in the design space, explore variations of their current design adjective, and the adjectives generated designs that were consistent with their expectations of the adjective (Figure 4.7). While most participants

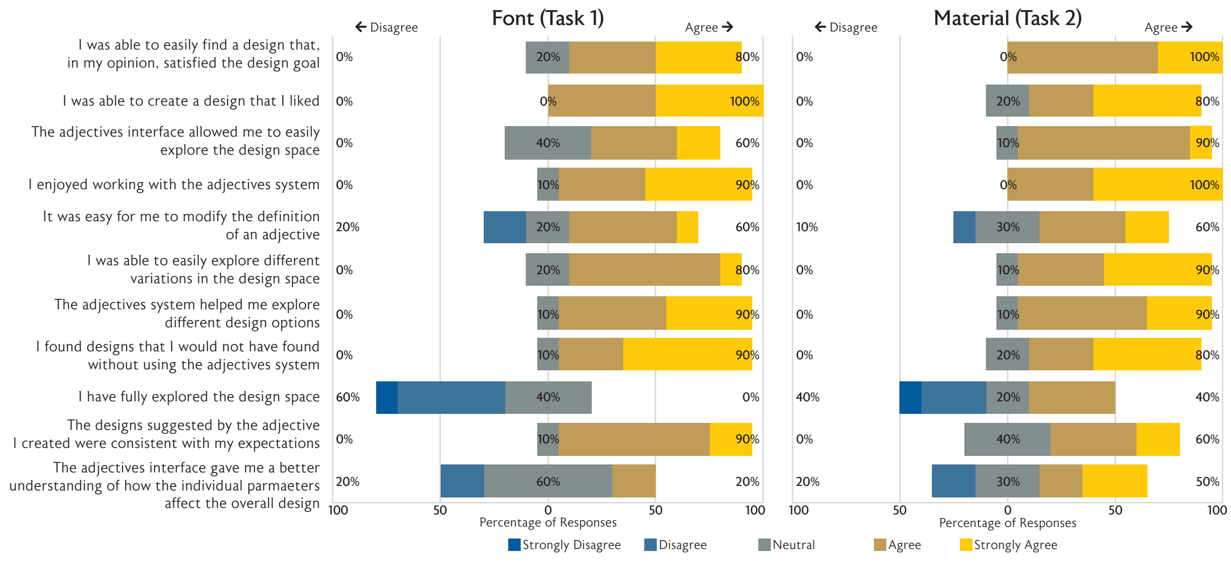


Figure 4.7: *Adjectives Interface Survey Results.* Likert plots of the responses to the survey questions about the adjectives interface used in the font design task (task 1, left) and the material design (task 2, right). Responses are generally consistent between tasks using the adjectives configuration.

agreed that the adjective definition was easy to modify, a few disagreed, possibly indicating that the initial examples provided to the adjective held too much weight. Participants used an average of 23 examples (13 positive, 10 negative) to define the "futuristic" adjective in Task 1, and 21 examples (11 positive, 10 negative) to define the "weathered" adjective in Task 2. In contrast to prior work [MM98, FF93], participants expressed no difficulty in locating negative examples. This is likely due to the preliminary random sampler outputting a large number of clearly irrelevant samples that can easily be marked as negative at the start of the process.

Participants agreed that the adjectives configuration allowed them to find designs that they might not have found otherwise. This suggests that the design suggestion sampling method for adjectives is effective at providing suggestions that are different enough from the input examples. Another source of this variation comes from the global samplers, however, participants indicated that they found the adjective sampling to be more useful (median score of 5) to them than the general random sampling operations (median score of 4) (Table 4.1). Analysis of the interface logs shows that users ran 2 global sampling operations on average and 5 adjective sampling operations.

Participants felt that they were more easily able to explore different variations in the design space with the design adjectives configuration when compared to the baseline sliders configuration. Plotting the designs explored in the space during Tasks 1 and 3 with a PCA projection along the first two components shows that users did indeed end up exploring many more dissimilar designs while using the design adjectives interface (Figure 4.11). While some users attempted to re-create the designs they made in task 1, other users indicated in verbal feedback that re-creation of the initial design they found was not one of their goals in task 3. Some participants noted that some of the designs found while using the adjectives configuration were interesting but orthogonal to their current design goal and wanted to use multiple adjectives to keep track of those designs. This capability exists in the interface, but was hidden for this study.

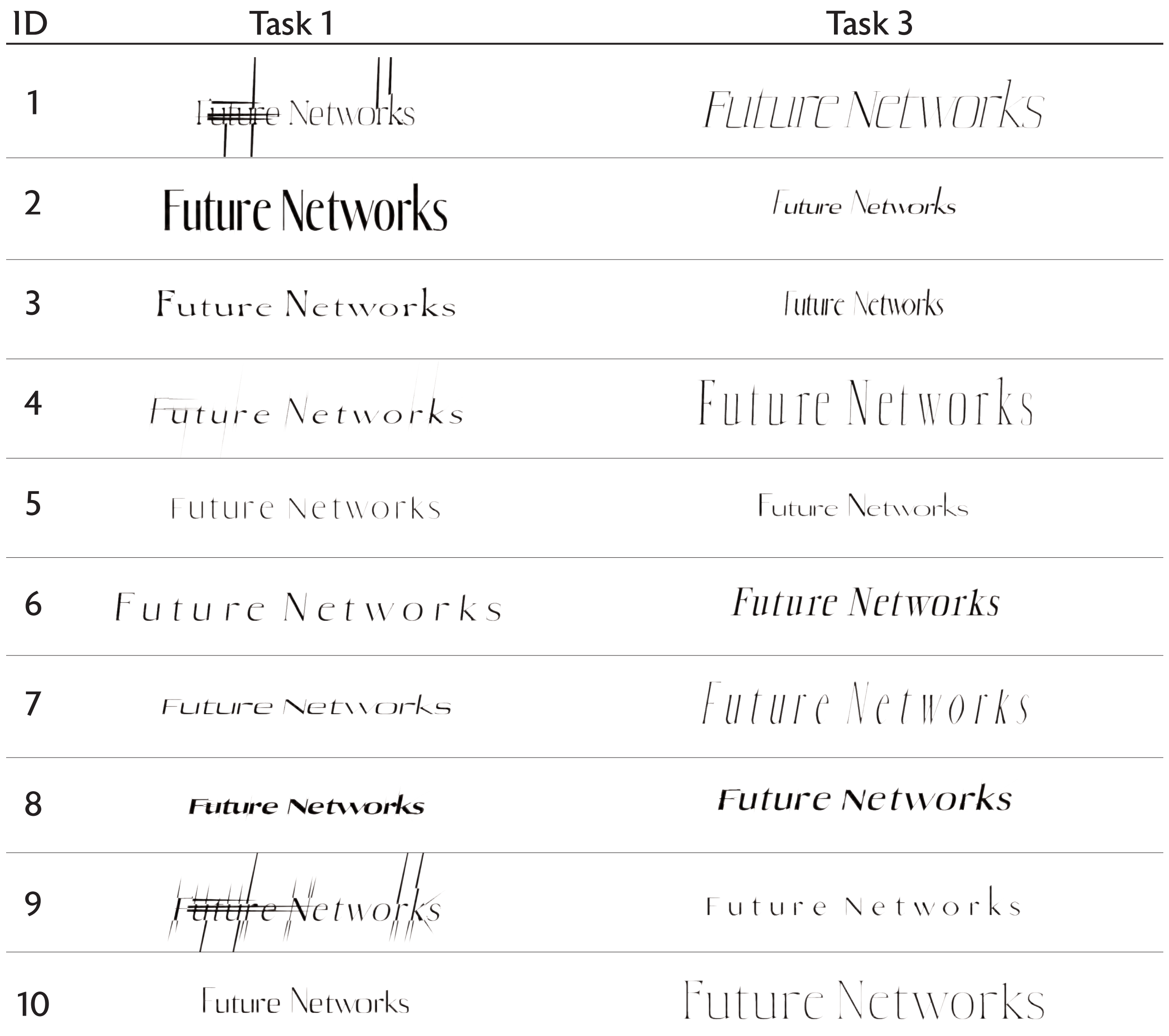


Figure 4.8: *User-Created Designs for Tasks 1 and 3* Results created by the user study participants for task 1 (left) using the adjectives interface configuration, and task 3 (right) using the sliders interface configuration. Many users commented that they actually liked the serif aberrations they found with the adjectives configuration, but were unable to re-create that effect with the sliders configuration.

Participants felt that adjectives configuration did not particularly help them understand how individual parameters affect the design space, however only half of the participants in the study ended up using the per-parameter controls (Table 4.1). Participants were still successful in creating a design that they liked with the adjectives configuration, indicating that the adjectives configuration can model user preferences accurately even without the user understanding how individual parameters affect the design.

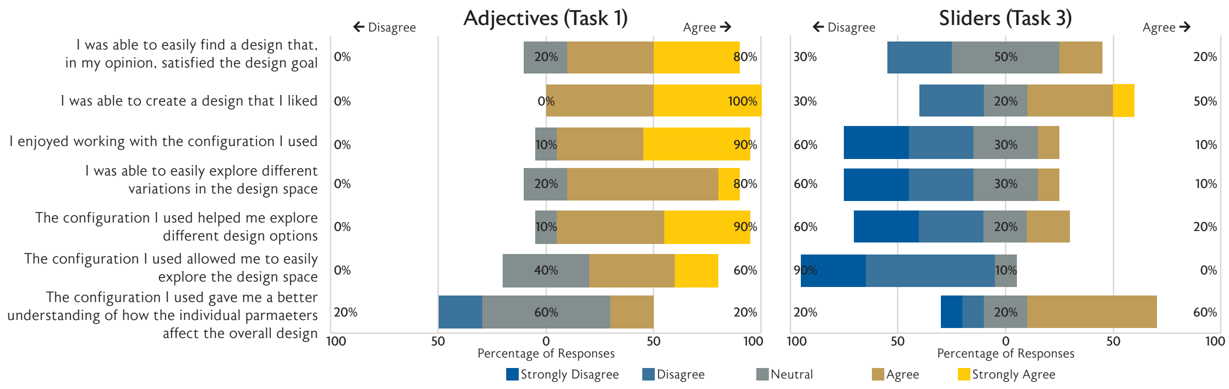


Figure 4.9: *Adjectives vs. Sliders Interface Survey Results.* Likert plots of the responses to the survey questions about the adjectives interface used in Tasks 1 and 3. Participants generally preferred to use the adjectives interface, despite Task 3 being a repeat of Task 1 with the sliders interface configuration.

	Task 1			Task 2		
	Median Rating	% Used	Avg. Use Time	Median Rating	% Used	Avg. Use Time
Adjective Sampler	5	100 %	322 s	5	100 %	259 s
Global Samplers	4	100 %	125 s	4	90 %	148 s
Parameter Highlighting	4	50 %	53 s	4	50 %	152 s
Parameter Selection	4	40 %	45 s	5	80 %	51 s
Parameter Extents	4.5	20 %	115 s	5	60 %	105 s
Mixer	5	50 %	97 s	4	50 %	94 s

Table 4.1: *Component Usefulness Scores.* Summary of survey feedback regarding the usefulness of each adjectives interface component. Participants were asked to rate how useful each component was on a five-point scale. The rating column reports the median score from these questions, the "% Used" column reports the percentage of participants that used the component at least once during the trial, and the "Avg. Use Time" column reports the average amount of time spent using the component in seconds (see Figure 4.10 for when each tool was used). Ratings and times are only taken into account if the participant used the component (validated by the trial's action log).

The Adjectives Configuration is Preferred for Exploratory Design Tasks Participants appeared to be more engaged with the adjectives configuration, reporting that they liked working with the adjectives configuration much more than the sliders configuration, while also spending less time with the sliders configuration task. Participants took 9.5 minutes to complete the font design task on average with the adjectives configuration and 6.3 minutes on average with the sliders configuration. The shorter completion time and the survey results in Figure 4.9 suggest that while users were able to complete all tasks in the study, they appeared to be more engaged with the adjectives configuration, to the point of voluntarily spending as much time as possible with the adjectives configuration. Whether or not users were searching for the same design that was created in task 1, there was a strong sense of frustration with the sliders configuration, with one user commenting that "using the sliders was not an enjoyable experience. While I felt I did not have enough time with the adjectives interface, I could not wait to be done with the sliders task."

Participants were generally able to complete the task to their satisfaction, although a higher percentage of participants were dissatisfied with their final results in Task 3. Their dislike of the sliders configuration was specific to its use for exploratory design tasks, noting that while the sliders limited the number of variations they felt they could explore, they did prefer to use the sliders in the detail design phase. One participant explained that "the slider interface was nice for fine control (somewhat slow), while adjectives was kind of fast. I think there could be a nice workflow going between these two configurations." The activity trace for this user (ID 2) in Figure 4.10 suggests that they used the adjectives configuration in this way, starting with adjective sampling operations and then using sliders for fine control.

When asked to pick one interface to use for each part of the design phase in a forced choice comparison, users again indicated a strong preference for using the adjectives configuration over the sliders configuration for exploratory tasks, with all participants stating that they would use it for general exploration ($p \ll 0.01$), and all but one participant stating that they would use it for directed exploration ($p < 0.05$). For performing detail-oriented design, a majority of participants (7 out of 10) indicated that the sliders interface was more appropriate for small, low-level design changes, with only 3 out of 10 users preferring to use the adjectives configuration for that phase of the design process. This last result is only weakly significant ($p > 0.05$).

Adjectives are the Most Commonly Used Component To better understand which parts of the interface contributed the most to the participant's evaluation of the adjectives configuration in Figure 4.7, each participant rated how useful they felt each component was, and we compare this data with how frequently each tool was used in the interface logs. Table 4.1 summarizes the results of this analysis, listing the per-task median score, the percentage of users who used the the component at least once during a task, and the average amount of time spent using the tool was during each task. We find that the adjective sampling was perceived as central to the adjectives configuration, as every participant used the adjective samplers at least once and rated it highly, while other framework components were seen as useful optional utilities. The bootstrapping tools (global random and jitter samplers) were also frequently used, but seen as less useful than the adjective samplers. The other components were used less frequently than both samplers, but were viewed favorably by the participants that did use those features.

The per-parameter features were rated highly, but not used as frequently as the sampling components. Many of these tools provide additional control over individual sliders, and some participants may have perceived these as unnecessary, as the adjectives alone were sufficient to accomplish their design goals. The usage rate for the per-parameter features (highlighting, selection, and extents) increased between Task 1 and Task 2. Since Task 2 contained more parameters than Task 1, many participants chose to perform operations that limited the number of active parameters (parameter selection). These tools, selection and viewing extents in particular, may be more useful as the number of parameters increases.

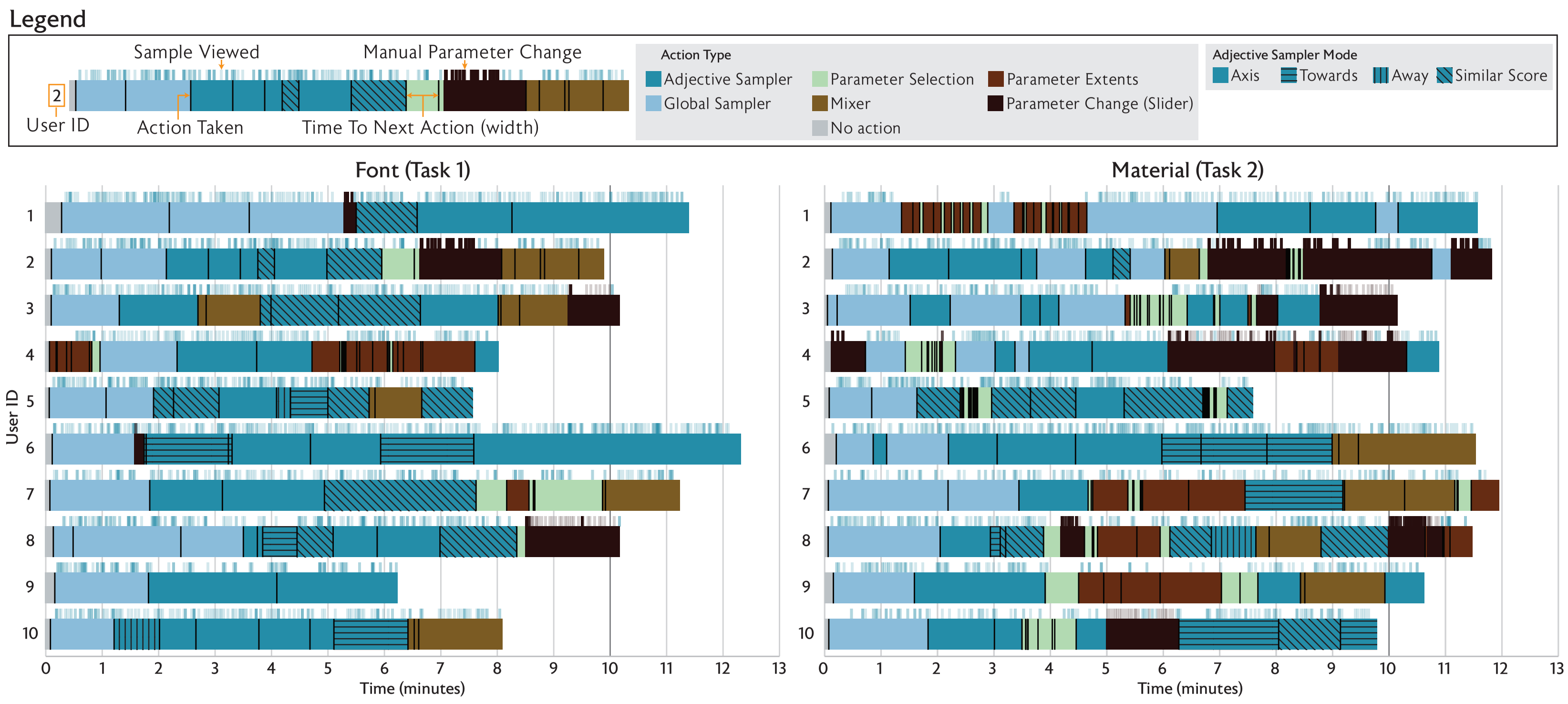


Figure 4.10: *User Activity Trace by Interaction Type.* Plot detailing when each user performing tasks 1 and 2 used an interface element belonging to one of the six component categories in the implementation. Each row of the plot displays one participant's actions over the course of the task. Each segment in a row represents when the participant activated an interface component, its width represents the time until they activated another component, and the color indicates the action type. The time between actions is typically when users inspect the results returned by invoking the interface action. The marks on top of each trace indicate when the user previewed a new design from a set of samples (blue) or directly manipulated the design with a slider (brown). The adjective sampling mode is indicated by the hatching pattern on the block.

Adjectives are Used for Directed Exploration To validate the claim that adjectives are primarily used for the exploratory phase of the design process, we plot the times at which participants used each interface component in Tasks 1 and 2, and for how long, in Figure 4.10. Each segment of a timeline marks when a major interface component has been invoked, and its width indicates the time until the next major component is used. The major components are the adjective sampler operations, global samplers (random and jitter), parameter selection, the mixer, and individual parameter changes (sliders). With one exception (user 4 in both tasks), the global samplers are all used first, followed by the adjective samplers, with a few selection operations sometimes happening before adjective use.

In both tasks, the activity traces reveal a design process that is closely aligned with the three conceptual phases of the design. Users generally performed preliminary design by using the global samplers, refined their design idea by using the adjective sampling methods, and often performed detail work with the individual parameter controls, extents view, or mixer components at the end. In general, we see usage of the design adjectives tools where we expected; comprising the bulk of the design process, used to refine and explore preliminary concepts. Participants used per-parameter controls less frequently in Task 1, possibly due to most parameter configurations outputting valid fonts. In contrast, Task 2's material had a larger possibility of outputting invalid designs, requiring users to use the parameter selection, extents, and direct controls more frequently.

Speed is Important Speed is critical in interactive design applications, and the adjectives configuration is no exception. While the sampling operation for adjectives completed within a second, the prototype interface required a few second to render the returned design suggestions. A few users expressed frustration with this small delay, commenting that "The biggest thing that threw me off is the rendering time/lag," and that "it takes a while to render and thus its harder for me to see the differences between similar things." Interfaces implementing design adjectives-style framework should ensure that the training and sampling of adjectives, and rendering of results, completes in real-time.

4.6.2 Professional Case Study

We asked a professional graphic designer to create a set of fonts using our implementation of the design adjectives framework. Many of this designer's projects involve the selection and editing of a font for a logo or package design. Their typical workflow involves the selection of a base font, followed by the generation of a number of design-specific adjustments to the base font, which are then presented to the client and further refined based on that feedback. The interface used to perform these adjustments is typically a glyph editor, where the font characters are edited directly.

The designer was given the design adjectives interface, five Prototypo [Pro19] font templates (Antique, Elzevir, Fell, Grotesk, and Spectral), and a 15-minute tutorial. The font templates contained 26 parameters controlling different font properties, without requiring the use of a glyph editor. While the full Prototypo interface offered glyph-level control in addition to parameter-level control, re-implementing a glyph editor in our interface was out of scope for this project. They used the interface unsupervised over the course of one week and created the 15 fonts in Figure 4.12. When working with the templates, the designer set out to create distinct fonts within each template. Each font took an estimated 16 minutes on average to create, as reported by the designer, and are considered by the

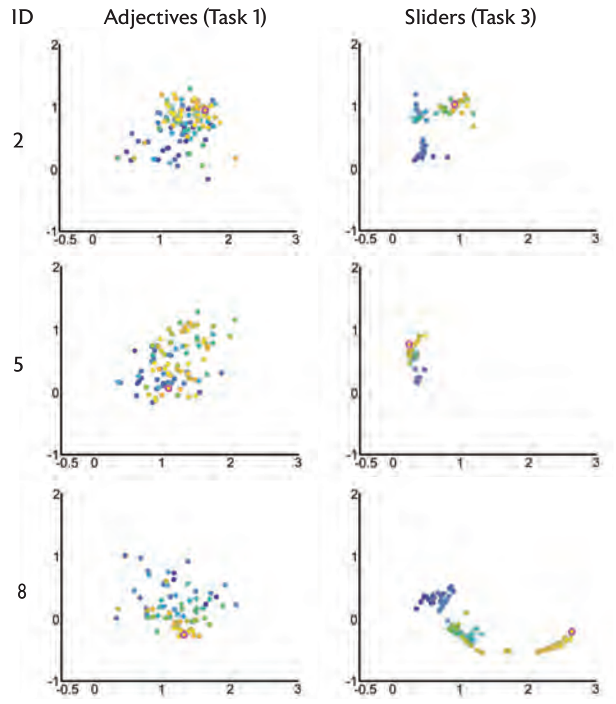


Figure 4.11: *2D PCA Projections of User Design Sessions.* Users tended to explore a more diverse set of designs while using the design adjectives configuration. These plots represent designs seen by participant IDs 2, 5, and 8 during design sessions for Tasks 1 and 3 projected into 2D along the first two PCA components. Color indicates when a design was seen during the process, with blue indicating earliest and yellow indicating latest. The pink outlined point is the final design chosen by the user. The same embedding is used for all plots.

designer to be usable in professional work with a few glyph-specific tweaks.

Overall, the designer enjoyed working with the tool, specifically noting that the tool was especially effective at providing a framework for quick design concepting. Their primary workflow when using the tool consisted of creating a new adjective, then adding one positive (high rated) example, and one negative (low rated) example to an adjective. The designer would then perform a "towards" sampling operation, update the current design, and repeat the addition of one positive and one negative example to the adjective. The designer would repeat this refinement cycle until sampling "towards" stopped returning results. This process converged in three to four cycles and "almost always produced an adjective that was good enough to start detailed refinement." In the instances when this method was not used, the designer bootstrapped the adjectives by generating a set of random samples, and then used the mixer tool to create examples. The designer noted that the random samples would often have features that would be considered unreasonable by most design standards, such as distorted serifs, and that the implementation may want to impose some stricter bounds for the parameters in this space.

Analysis of the interface's recorded logs verify the designer's assertions. The logs show that the professional designer typically followed a usage pattern similar to that of the intermediate designers in Figure 4.10. They first bootstrapped the adjectives using the global sampler, ran the adjective

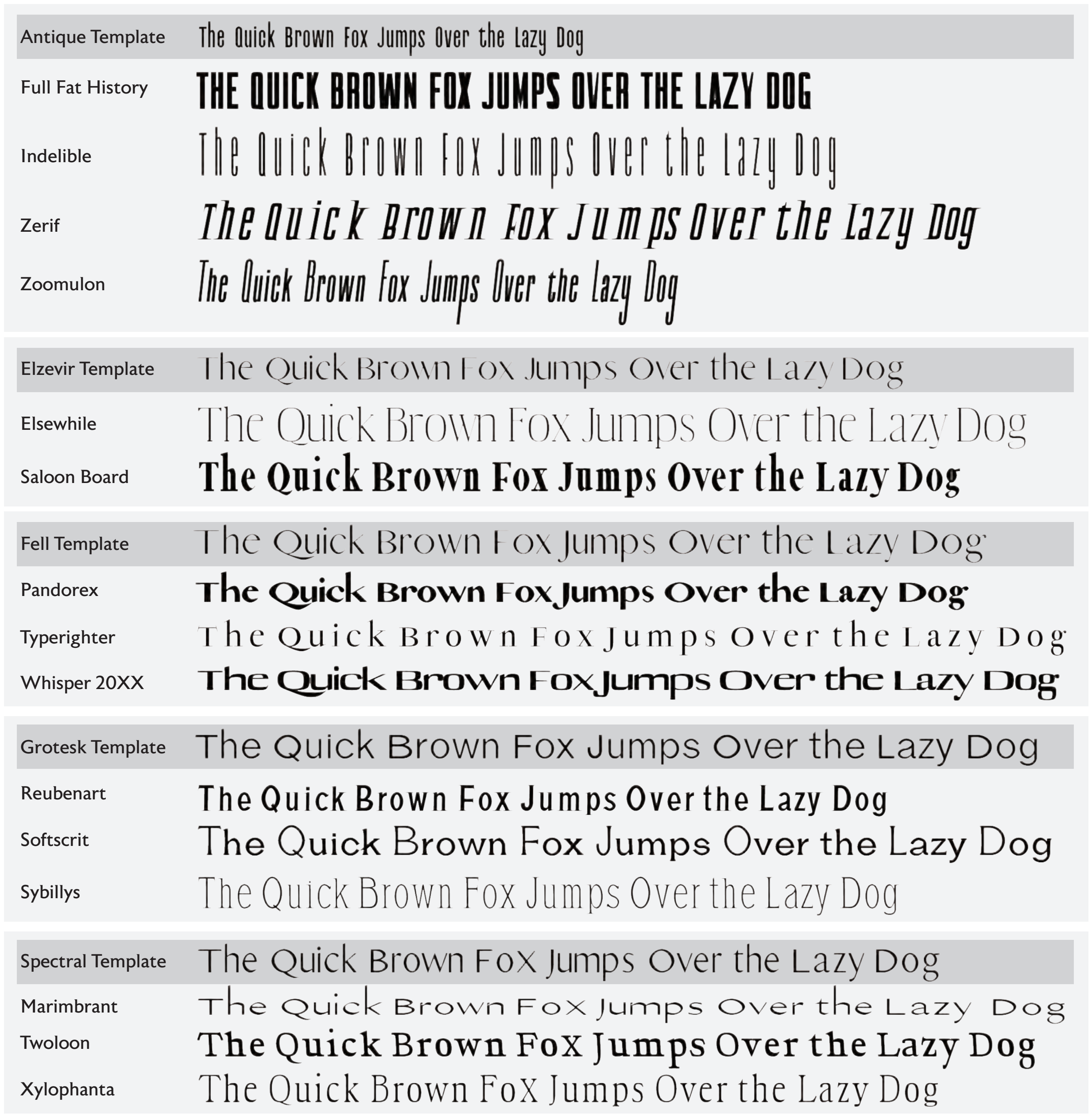


Figure 4.12: *Professionally Created Fonts.* 15 fonts created by a professional graphic designer using the design adjectives interface. Fonts are grouped by which Prototypo template was used to create them, with the designer chosen font name displayed on the left. The template font is shown at the top of each group.

sampler a few times, and finished designs with the mixer and per-parameter tools. This professional liked using the mixer, and the log shows frequent use of that tool at all parts of the design process.

One of the limitations of the tools in the designer's view was the lack of a sideboard, a panel in the interface that can store designs unrelated to any current adjectives for later use. To get around this limitation, the designer created a “sideboard” adjective, where the scores did not matter, to store these designs. The designer expressed some frustration with the tool’s parameter selection capabilities, mentioning that it was difficult to find the controls for preventing a parameter value from changing during sampling. While this capability exists in the implementation, it was not presented clearly enough for the designer to make effective use of it. Fonts created in this interface could be exported to existing glyph editing tools for final adjustments, after the design concepts have been created. Parametric fonts such as those used in this case study already provide a high-level interface for modifying all of the font glyphs at once, without resorting to a low-level glyph editor. Even in this context, the designer felt that the adjectives framework helped them locate interesting variations much more quickly than directly manipulating parameters.

4.6.3 Material Design Session

An professional material designer spent a day experimenting with our implementation, creating and adjective and twenty representative variations of that adjective for a brick texture in Figure 4.13. The adjective (left unnamed by the designer) used for this design session included 160 labeled examples. Figure 4.13 includes visualizations of the axis and towards sampling acceptance criteria, and details an professional’s thought process regarding how a design gets refined by using sampling results combined with the mixer interface component. High resolution images of the created design variations and the professional’s process diagrams can be found in the supplemental material.

4.7 Discussion

This section presented an implementation and evaluation of design adjectives, a software framework for building parameterized design interfaces that guide users using learned models of intent. We found that when using tools built on top of this framework, users were more easily able to create designs that satisfied a design problem according to their personal preferences. We hope that the framework, and our domain-agnostic implementation of its components, serves as an inspiration and baseline for the creation of new component implementations and new design tools that can react in real time to the ever changing preferences of designers as they solve ill-structured design problems.

4.7.1 Limitations

The design adjectives sampler requires at least one positive example in order for the guided sampler to run, and creating this initial positive example can be difficult. The global randomization bootstrapper works well enough for the design spaces used in the evaluation, but may struggle to provide valid examples in spaces where only a small number of parameter configurations produce a satisfactory initial design. The GPR formulation of adjectives permits use of more complex priors to help shape the adjective function, but creating these priors may require expert knowledge about a specific design space. For example, it may be possible to create classifiers that mark obviously bad designs (e.g. for a material: overexposed, low contrast, etc.) in a space and then bootstrap by presenting the user only examples that pass this filter.

We chose to focus on workflows involving a single design adjective in order to demonstrate that adjectives can represent design concepts defined by one user. This is an important baseline to establish before adjectives are used as the basis for more complex interfaces. It is possible that a user might create a library of adjectives and wish to use multiple adjectives at the same time during a design task. Understanding such use cases was outside the scope of this project.

Adjectives are only valid for the specific design domain they are learned on. For example, if an adjective is defined on a brick material created with Substance3D, the adjective cannot be used on a brick material that has a different parameterization. Since our framework only operates on the low-level parameters, this type of domain transfer of intent cannot be implemented. It may be possible to create transferrable design adjectives if the adjective has knowledge of the visual properties of the design. This type of understanding may also enable better per-parameter identification tools, as the framework would have a way to recognize what parameters affect specific visual properties.

Continuously sampling from adjectives is an efficient means of moving across the space, but encounters difficulties with preserving per-parameter edits made by a user. In the current implementation of design adjectives, these edits can be preserved by excluding the parameter from future sampling operations. However, it may be more desireable to keep those parameters involved in sampling, but still retain characteristics of the original edits.

Additionally, more evaluations of interfaces created with the design adjectives system will be required in order to fully understand the support provided by the framework. The evaluation in this chapter focused on self-reported metrics and how individual users felt while using the interface, and indicates that users feel that the system provides some benefit. Future studies may want to investigate to what extent the design space is explored before finding a design that meets a set of specific criteria set out in a design specification, how users interact with pre-made adjectives versus creating their own adjectives, and what the full extent of the perceived benefits of the system are. In that regard, future studies will have to establish a stronger sliders interface baseline, as the study in this chapter utilized a repeated task, with some users choosing to continue to explore the design space with the sliders and other attempting to re-create the design they found in the first task.

4.7.2 Future Work

We believe that our framework opens exciting avenues for future work. Our approach could be applied to local editing could enable a new take on the creation of complex structured artifacts, e.g., to manipulate the materials of a large 3D scene. The framework could be extended to a collaborative environment where multiple users who each have their own collections of adjectives could combine their adjectives into a library, creating opportunities to share information and combine adjectives. Finally, we also see a lot of potential between our framework and other design metaphors like sketching and image-based modeling, that could be used to create better bootstrappers for a design adjective. We hope that providing an implementation of this framework will makes it easier to create and experiment with these future interfaces. These future work ideas are discussed in more detail in Chapter 7.

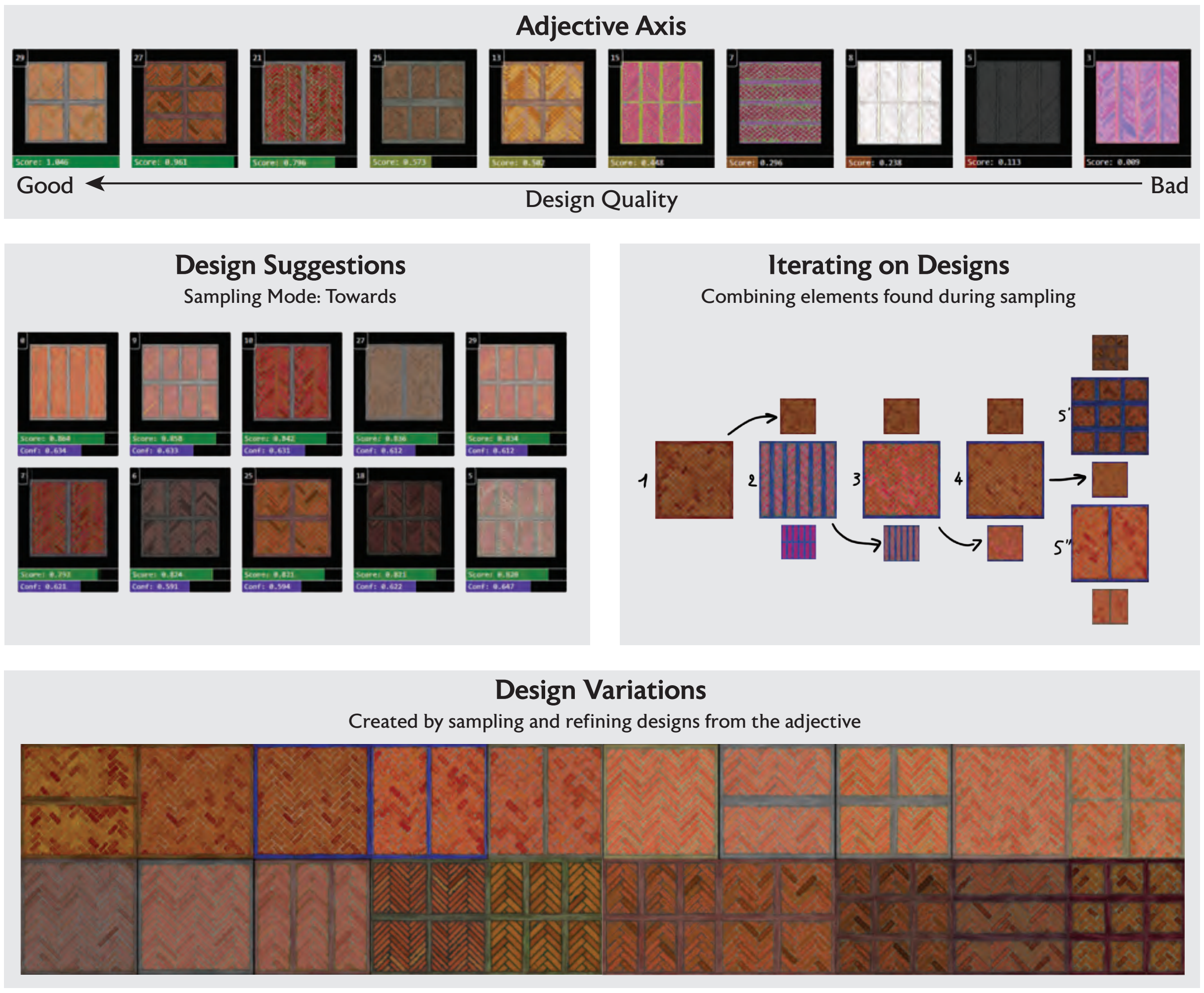


Figure 4.13: *Professionally Created Materials.* A visualization of a professional design session with the adjectives interface. The adjective definition is visualized by displaying the results of sampling using the Axis acceptance criteria (top). Samples generated by using the Towards criteria produce designs that are highly rated by the adjective (middle left). These designs can be iterated on by mixing together pairs of sampled designs (middle-right), where the large design is created from a combination of the two smaller designs on the top and bottom. A set of 20 designs created from this adjective is shown on the bottom.

Chapter 5

Bootstrapping Theatrical Lighting Design using Visual Objectives

The design adjectives system introduced in the previous chapter focused on supporting the iterative design loop of refining an idea over time, but relied on a uniform random sampling of the design space to bootstrap that process. While this approach works for design spaces where random sampling generates enough plausible results to help the designer get started, it is less likely to be successful in domains where there are fewer valid configurations. Creating a better preliminary design generation method is difficult in generic spaces, as there are no design principles to fall back on, but should be possible in domains with strong, well-established principles. This chapter takes a look at one such domain, theatrical lighting design, and describes the design principles and the implementation of a preliminary design generator derived from domain-specific principles.

Portions of this chapter previously appeared in the paper "Exploratory Stage Lighting Design using Visual Objectives" [SPF$^+$19]. As this work was completed before the design adjectives framework was developed, this section describes the "visual objectives system,“ which is set of modeling, sampling, and rendering components that could be used as part of a design adjective framework implementation. The relationship between design adjectives and visual objectives will be discussed in Section 5.7.

5.1 An Introduction to Theatrical Lighting Design

In order to understand how we can customize design adjectives to better suit theatrical lighting design, we will need to have a basic understanding of what theatrical lighting design is and how it differs from lighting in virtual contexts. In theater, lighting design is a critical part of creating compelling imagery. Illumination provides cues about time of day, environment (e.g., the blue sheen of moonlight, the uniformly gray look of a rainy day), and mood or emotion (e.g., a mix of bright colors evokes a festive scene, purples and pinks can look romantic). Per-parameter controls are used to create lighting designs in modern stage productions under tight time constraints, even in shows featuring tens to hundreds of colored light sources with limited pre-visualization support.

Although many efforts have explored computational techniques for crafting or altering illumi-

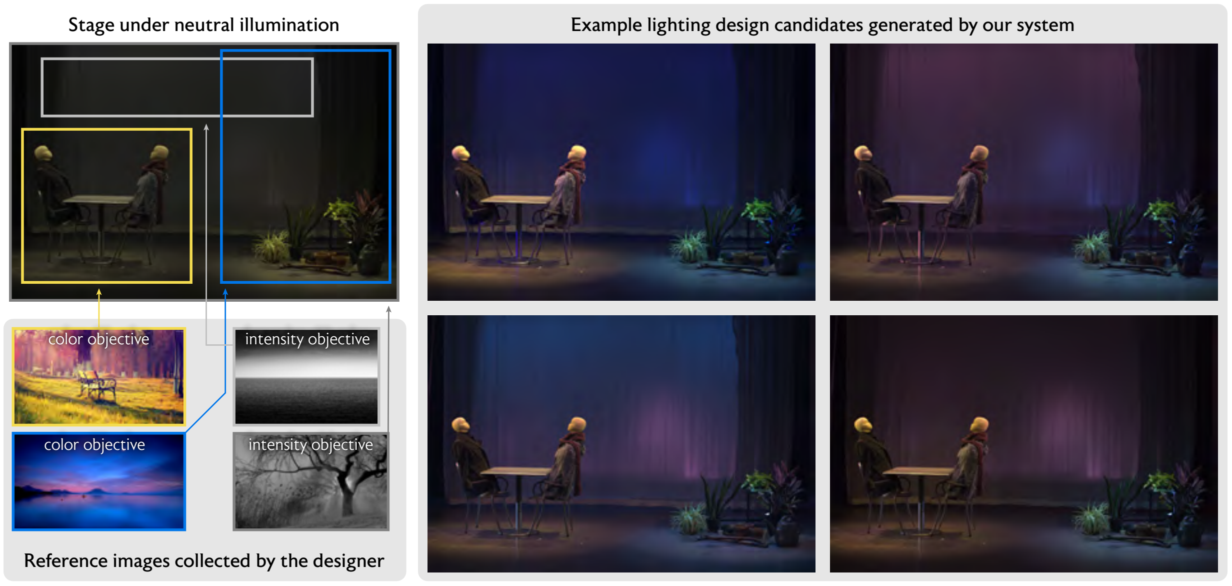

Stage under neutral illumination
Example lighting design candidates generated by our system
color objective
intensity objective
color objective
intensity objective
Reference images collected by the designer

Figure 5.1: We introduce a system that aids exploratory theatrical lighting design. Designers abstractly express intent in the form of reference images, and select aspects of the images (e.g., color or intensity) to apply to regions of the stage. From these visual objectives the system generates a gallery of design candidates. Designers can explore and refine lighting designs by adding or removing visual objectives.

nation of virtual 3D scenes or digital images, in discussions with professional lighting designers it became apparent that theatrical scenarios present different challenges and workflows compared to virtual lighting. In theatrical lighting, designs must be physically realizable on a stage, preventing many powerful editing paradigms (e.g., compositing) that are possible when manipulating lighting in photos. In contrast to portrait or product photography, where positioning and choosing fixtures is a key part of the design process, stage lighting often is constrained to a fixed (or limited set) of light positions and angles dictated by the physical structures present in the theater space. Most notably, the early stage of theatrical lighting design is an abstract and exploratory process. As one designer told us, designers seek to establish creative illumination environments that often "are not like what you see in the real world." Theatrical lighting evokes a sense of place and mood with exaggerated colors and varied lighting angles. To create these effects, designers often do visual research: they collect images to reference (of abstract artwork, dramatic photographs, etc.) as they try to combine elements from these images on stage to achieve a desired look, tone, or feel.

Theatrical lighting design follows the general structure of a parameterized design problem outlined in Section 2; preliminary designs are presented to a director, and the ones that are approved get refined and detailed on a physical stage. Based on discussions with professional lighting designers, we determined that the detail design phase was well supported while the preliminary and refinement stages were not. To address this shortcoming of existing lighting control systems, we created a gallery-based interface [MRR^{+}97] that generates design suggestions by extracting a model of the desired lighting that captures the visual feel (color palette, intensity, contrast) from a designer-provided reference image. This model is called a *visual objective*. To generate designs from the visual objective model at interactive rates, we use a Gibbs-sampling [CG92]-like approach that adheres to common theatrical lighting design principles. Users can then select designs to refine, mixing and matching aspects of their visual research in order to rapidly explore different design choices. The resulting designs are physically realizable, and can be transferred to a real stage.

While many of the techniques employed in the system (image-based relighting, design galleries, sampling-based design generation) are inspired by prior work in computer graphics, this project contributes an interface specifically targeted at the requirements for exploratory design in theatrical lighting design workflows. The system is validated through a series of user studies, in which we find that designers are able to quickly create good starting points for complex designs and can use these preliminary designs to more effectively communicate their ideas to other designers or directors.

5.2 Prior Work

Visual objectives extend design adjectives by providing a bootstrapper for theatrical lighting design. Instead randomly generating designs, the initial gallery is populated by lighting designs generated from a model of theatrical lighting design, which uses images to describe lighting properties. Sampling the entire lighting design space is intractable, instead, this method creates a model from statistical properties of images and samples designs from it using a Gibbs-like sampling method. The statistics captured do not fully describe the design space, for example we do not estimate lighting direction [LMGH$^+$13], as the images do not necessarily represent realistic lighting. The model utilizes theatrical lighting design principles in order to provide viable designs in seconds, instead of attempting specific goal-based optimization.

We use re-lighting techniques to render high-quality visualizations in real-time. Compositing-based methods for editing images [ALK$^+$03, ADA$^+$04, BPB13] provide the freedom to create lighting conditions not possible in the real world. Unfortunately, we cannot selectively pick which pixels should be rendered in the real-world, preventing the use of sophisticated compositing in theatrical scenarios where the output must be realizable on a physical stage. Therefore, we use compositing for visualization and limit it to physically accurate linear blending [PVL$^+$05]. The fast real-time rendering techniques in [DAG95] could be implemented in our system to provide better support for animated effects, however this is not the focus of the work.

In comparison to existing lighting design systems, the visual objectives system seeks to help practitioners set the overall tone and feel of a scene via image examples. Some prior work gives the designer direct control over key lighting features, such as the placement of highlights or shadows [PF92, PTG02]. Others offer painting-based interfaces where an artist directly specifies target pixel values, allowing the system to determine which lights are able to create the specified effect [SDS$^+$93, ADW04, PBMF07]. Both lighting feature manipulation and paint-based systems, as well as other goal-based systems (e.g., ensure a region of the scene is adequately lit [KPC93, SW14, KP09]) seek to optimize low-level lighting parameters to meet objectives that fully define the desired output at specific locations, e.g., "put a shadow here" or "add a highlight there".

Since the early work of Dorsey et al. [DSG91] few computer graphics researchers have targeted work specifically at the needs of theatrical lighting. While growing complexity of stage lighting configurations (including the introduction of color changing LED lighting) has led to increased interest in software visualization and authoring tools, leading commercial systems remain based on slider-per-light control interfaces and visualization tools that lack photorealistic rendering support [L8 , MA , Vec20, Sof]. These tools are best suited for addressing timing and light beam animation concerns of stage spectacular or concert lighting, not assisting a designer with subtle color and intensity choices needed to evoke tone or mood.

5.3 Observations and Design Principles

The first phase of our project involved extensive discussions with theatrical lighting design professionals about their creative process. We describe this process below, then identify the key principles that guide the design of our system.

In the preliminary phase of a lighting design project, it is common for designers to collect images that represent potential design ideas. For instance, a painting may be used to specify a color palette and a movie still for light intensity and contrast. These *reference images* are used throughout the design process to communicate lighting design ideas to the rest of the design team (e.g., other designers or a director). Examples of reference images are shown in Figures 5.2-right, 5.3-left, and 5.10-bottom.

Using the gathered reference images, designers then try to visualize their ideas on the actual stage to evaluate the color, contrast, and mood created by the lighting configuration. However, due to the lack of high-quality visualization tools, most design refinement occurs only after access to the physical stage is available. This forces designers to rapidly iterate over their designs under tight time constraints, accepting minor inaccuracies in the design until the entire show has been roughed in. During this process, designers first work to establish the overall look of the scene, adjusting the global intensity distribution and color palette. Once the general look has been set, the designer may continue to adjust individual lights as needed to perfect the design.

Lighting designers deal with tens or hundreds of light sources, and significant experience is required to organize these devices in a way that allows for them to realize their ideas on stage. The majority of light sources used in a theatrical setting are static and cannot change where they point, or their beam properties after being placed. The primary way of dealing with this complexity is to organize lights with the same "function," such as "front light" or "side light" into *light groups*, and use the same setting for all lights in a group. Designers often call light groups "systems" but we avoid this term in this thesis for clarity. Although individual lights may be adjusted during later refinement, light group granularity manipulation is an effective way to achieve results quickly during the exploration phase. Light groups are created by the designer as they place lighting fixtures in the available locations around the stage.

In order to work with current lighting control interfaces, lighting designers must specify brightness in terms of light intensity, which is the percentage of the total luminous output of a lighting device. When specifying the intensity of a scene, designers select some light groups, called *key lights*, to highlight the main elements of the scene. Key lights are the brightest light groups in the scene. The remaining groups, called *fill lights*, are used for illuminating the rest of the stage and determine scene contrast. (Fill lights "fill in" the shadows created by key lights.) Designers consider the contrast of a design to be the intensity ratio between these two groups of lights (how much brighter the key

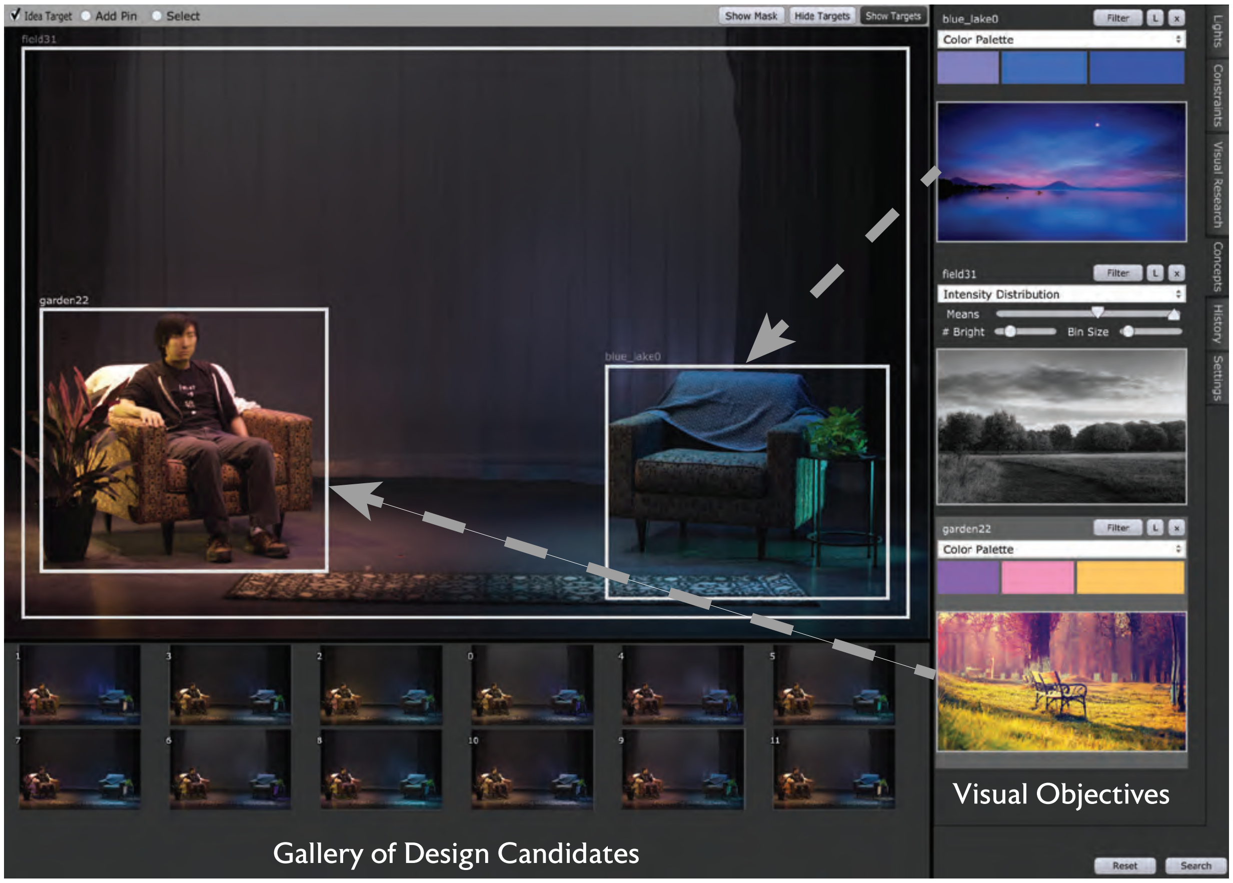


Figure 5.2: *User interface.* The visual concepts interface takes the current light parameters, and a list of visual objectives (right panel), and generates design candidates that satisfy these objectives (bottom panel). The user can localize each objective to a different region of the stage (boxes on top left panel). In this case, the two color objectives apply to the marked boxes, while the intensity objective (greyscale image) applies to the entire stage. The process is iterative; the user can move a candidate to the stage and assign new objectives which will only modify the targeted regions' light color or intensity.

light regions are compared to the rest of the stage).

The description of the lighting design process presented here should be reminiscent of the general parameterized design process discussed throughout this thesis. In this domain, we are interested in providing a method for more quickly bootstrapping the design process, as randomly setting light colors and intensities is not likely to provide valid designs to work with. In order to implement this bootstrapper, we are going to use some domain principles and observations about the lighting design workflow. In particular, this bootstrapper will need to:

Interpret Designer Intent in Images Designers should be able to rapidly communicate desired visual properties of lighting to the system using images, much like how they communicate ideas to other designers and directors.

Adhere to Theatrical Design Principles The system should model theatrical lighting design principles so that generated designs are plausible and visually attractive. For example, it should understand relationships between key and fill lights and respect the designer-provided light groups.

Allow Partial Design Specifications In cases where a designer has specific properties in mind, generated designs should adhere to those constraints. For example, a designer may only wish the system to generate changes to a specific stage region, leaving the rest of the stage unchanged.

5.4 Method

In this section, we describe a system for stage lighting design that is based on the observations, principles, and conversations with experts described in Section 5.3. Selecting light positions is not typically a time-constrained activity in lighting design, so we assume that the position and angle of scene light sources has already been specified (a common setup for theatrical lighting design scenes). After selecting light positions, the task is then to assign colors and intensities to each light for each scene under a tight time constraints. We will refer to an assignment of colors and intensities to all lights as a *lighting configuration*. We first provide an overview of how a designer works with the interface to specify design ideas and refine system-generated design candidates that embody these concepts. We then describe the key algorithmic details of the system's implementation.

5.4.1 System Overview

Figure 5.2 shows a screenshot of the visual objectives interface. At the beginning of a work session, the designer imports reference images that embody aspects of the tone, feel, and visual appearance they seek to recreate on stage. We refer to these images, along with a specification of which characteristics of the image the designer finds desirable (color palette, intensity distribution, or both), as *visual objectives*. Figure 5.2-right shows two visual objectives that suggest desired color palettes, and one that suggests overall scene intensity and contrast.

Given a set of lights on stage, an organization of those lights into groups, and a set of one or more visual objectives, the system generates a set of lighting configurations, which we call *design candidates*, that embody the objectives. When generating design candidates, the system respects standard theatrical lighting design principles (e.g. key and fill lights) to generate a diverse set of designs. Similar candidates are clustered and presented to the user in the form of a design gallery (Figure 5.2-bottom).

From this example gallery, the designer can select an appealing candidate and refine it. Note that this interface is not a full implementation of design adjectives, as the methods for refining the candidate are limited to replacing the visual objective, adding new constraints to the objective, or directly manipulating light parameter values. These restrictions are discussed in more detail at the end of the chapter (Section 5.7). Examples of constraints include specifying a stage region visual objective should be applied to, or specifying what light groups to use as key lights. In Figure 5.2, the designer targets the color palette from the blue lake reference image to the right side of the stage, and the warm color palette from the outdoor meadow image to the left side of the stage. The middle image (landscape scene) is used to set the lighting intensity of the scene.

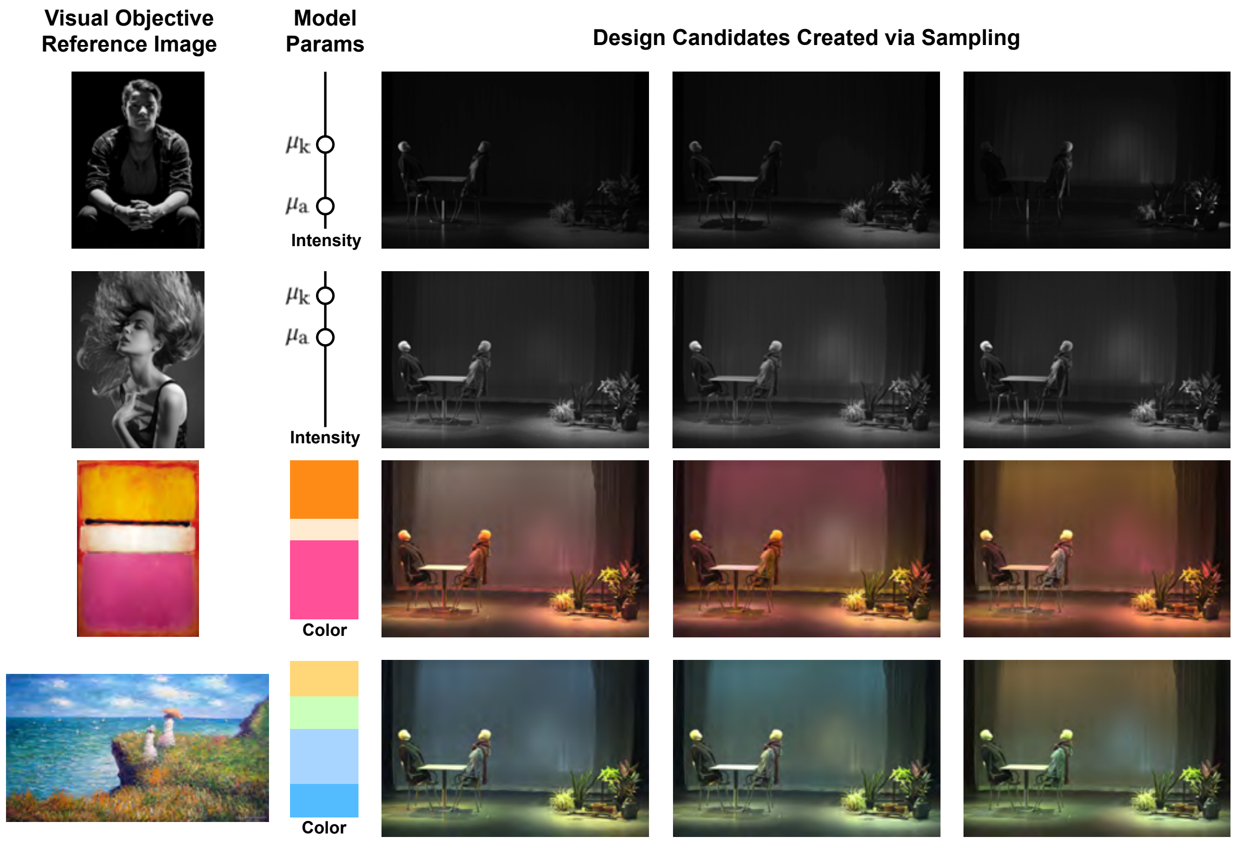

Visual Objective Reference Image
Model Params
Design Candidates Created via Sampling
μ_k
μ_a
Intensity
μ_k
μ_a
Intensity
Color
Color

Figure 5.3: *Intensity and color visual objectives.* In the *Intensity* model, we sample light intensities using a model based on the image's per-pixel intensities. In the *Color* model, we sample light color to match the image's color palette. In these examples, the visual objective is targeted to the entire stage.

5.4.2 Intensity Visual Objective

Since the primary goal of visual objectives is to aid experimentation with the placement and intensity of color on stage, we provide designers with two types of objectives: *intensity* and *color*. Designers express their intent by providing a reference image, which may be a painting, abstract pattern, or photograph that inspires them. Since the content (and light sources) of reference images often bear little resemblance to the target stage, it is not reasonable to assume a reference image results from physically accurate light transport that, if inverted, would yield accurate stage lighting parameters. For the same reason, an image-based optimization approach is also not desirable. Instead, the system extracts a model of the visual objective from the image, then generates design candidates by sampling from the extracted model. For simplicity, we first describe our model for the intensity visual objective and how it is used determine light intensities. We delay description of the color visual objective to Section 5.4.3.

Defining the Intensity Visual Objective

When describing the lighting intensity of a scene, lighting designers consider two factors: the scene's overall brightness, as well as its contrast (brightness differences between key and fill lights). As shown in Figure 5.4-top, when sampling light source intensities based on image histograms, most design candidates poorly match the average intensity of the visual objective.

Echoing how lighting designers think about brightness in terms of key and fill light intensities, our model of lighting intensity is based on two parameters extracted from the reference image. Most reference images are low dynamic range images, so to approximate lighting intensities, we convert the pixels to the CIELAB color space and treat the L* component as the intensity of the pixel. The visual objective's average intensity, μ_a, is given by the average pixel intensity of the reference image. The average intensity of key lights, μ_k, is estimated as the average intensity of the top 15% of pixels ordered by L value in the image (these pixels are assumed to be illuminated by key lights). Figure 5.3 provides two examples of intensity parameters.

Although the system assumes that light sources are already positioned (both location and angle), stages contain many lights, so exploration of lighting angle is possible by turning lights from a particular position or direction on or off. Therefore, while design candidates should accurately reflect the average overall intensity and key light intensity of the visual objective, it is also desirable for a sampling scheme to produce high variance in individual light intensities across candidates in order to produce a diverse set of candidates.

We find that an approach that is similar to Gibbs sampling [CG92], where each light intensity is conditioned on prior assigned light intensities, is able to generate design candidates that meet both of these goals. The method naturally extends to incorporate additional design constraints that may be optionally supplied by the designer (see Section 5.4.4), and can produce large numbers of candidates at interactive rates.

Generating Design Candidates with Gibbs Sampling

For simplicity, we introduce our approach to sampling light intensities assuming that one fourth of a stage's lights are key lights and that all lights on stage have the same maximum intensity. It is common for lighting designs to modify the key to fill light ratio, and we subsequently relax both of these assumptions in Section 5.4.2.

Given a stage with L lights, the sampler randomly selects $L/4$ lights to serve as key lights (uniform distribution). Starting with the key lights in random order, the intensity for the key light is drawn from $\mathcal{N}(\mu_\mathrm{k}, \sigma_\mathrm{k})$, where σ_k is a tunable parameter (5% of the available intensity range by default, a threshold used by lighting designers as the just noticeable difference). To match the average light intensity specified by the visual objective (μ_a), the algorithm sets the intensity of the fill lights, in random order, one-by-one, adjusting the distribution for the remaining lights as each step. Specifically, after the j^{th} light intensity has been set, the goal is to shift μ_{j+1} for the $(j+1)^{\mathrm{th}}$ fill light so that, if all remaining sources were sampled from this new distribution, the expected average across all lights is μ_a. That is, we define μ_{j+1} so that:

$$\frac{1}{L}\left(\underbrace{\sum_{i=1}^{L/4+j} x_i}_{\substack{\text{lights set}\\\text{so far}}} + \underbrace{\left(\frac{3L}{4} - j\right) \mu_{j+1}}_{\substack{\text{expected intensity}\\\text{of remaining lights}}} \right) = \mu_\mathrm{a} \tag{5.1}$$

where x_i's represent previously sampled intensities, including the key lights. Rearranging terms yields:

$$\mu_{j+1} = \frac{1}{3/4 - j/L} \left(\mu_{\mathrm{a}} - \frac{1}{L} \sum_{i=1}^{L/4+j} x_i \right) \tag{5.2}$$

Sample x_{j+1} is then sampled from $\mathcal{N}(\mu_{j+1}, \sigma_{\mathrm{a}})$, where σ_{a} is a tunable parameter (10% of the intensity range by default, fill lights typically vary more than key lights) and determine μ_{j+2} according to the value of x_{j+1}. Intuitively, this formula progressively adjusts the intensity of the light sources until the desired average intensity is reached. Formally, it amounts to sampling the distribution:

$$P(x) = \frac{1}{4}\mathcal{N}(\mu_{\mathrm{k}}, \sigma_{\mathrm{k}}) + \frac{1}{L} \sum_{j=1+L/4}^{L} \mathcal{N}(\mu_j, \sigma_{\mathrm{a}}) \tag{5.3}$$

where the μ_j parameters are computed as previously described. This procedure ensures that on average, key light sources have intensity μ_{k} and all the lights considered together have average intensity μ_{a}. Unlike a standard Gibbs sampler, our approach is not dependent on prior design candidates, and thus does not require a burn-in period. We provide pseudocode in the supplemental material.

Properties of Gibbs Sampling

Given our decision to represent the intensity objectives in terms of key light and overall intensity, we also considered directly sampling light intensities from a bi-modal distribution featuring a mode for the key lights and a mode for the fill lights. Specifically:

$$P_{\mathrm{BM}}(x) = \frac{1}{4}\mathcal{N}(\mu_{\mathrm{k}}, \sigma_{\mathrm{k}}) + \frac{3}{4}\mathcal{N}(\mu_x, \sigma_{\mathrm{a}})$$

where μ_x is computed as in Equation 5.2, but only once for the first $(j+1)^{\mathrm{th}}$ fill light. We found the Gibbs-sampling approach to be preferable for two reasons: it generates design candidates that are both more consistent with the visual objective (matches average intensity and contrast) and exhibit more variation across candidates (encourages exploration).

Consistent Average Intensity Gibbs sampling consistently generates candidates whose average intensity matches the intensity visual objective because the sampled distribution is sequentially adapted so that the expected average μ_{a} is conditioned on the previous samples' values. If the average intensity of the previous intensity samples is "off-target", the distribution is adjusted so that the expected average intensity of the entire lighting configuration is back "on-target". Formally, since $P_{\mathrm{BM}}(x)$ is an average of independent random variables, the variance of the output is σ_{a}^2/L. In contrast, the variance in Gibbs sampling comes from only the last sample, and is $\sigma_{\mathrm{a}}^2/L^2$.

Greater Per-Light Intensity Variance. While Gibbs sampling reduces variation in overall intensity of design candidates, it increases variance in the intensity values assigned to individual lights. As shown in the middle and bottom rows of Figure 5.4, this leads to candidates with higher contrast between light intensities and greater diversity in lighting direction. Ignoring the key light sources (since they are treated similarly in Gibbs and bi-modal distribution sampling), the variance in intensity values of any specific fill light source ℓ across design candidates is σ_{a}^2 for direct bi-modal distribution sampling and $\sigma_{\mathrm{a}}^2 + \mathrm{Var}_c[\mu_{r_c(\ell)}]$ for Gibbs sampling, where $\mathrm{Var}_j[\cdot]$ is the variance operator across candidates, $r_c(\ell)$ the random index assigned to the light ℓ when generating the candidate c, and $\mu_{r_c(\ell)}$ is computed via Equation 5.2.

Generalizing the Sampling Procedure

The proceeding sections assumed that $1/4$ of the lights in a lighting configuration were key lights. In practice, this results in candidates that are too dark for bright reference images (and vice versa for low intensity images). We address this issue by determining the number of key lights in a lighting configuration from the intensity of the reference image. We compute the proportion b of pixels in the reference image brighter than the mean image intensity, and then perform Gibbs sampling using $L_{\mathrm{k}} = bL/2$ light sources as key lights. This choice ensures that when 50% of the pixels in the reference image are brighter than its mean intensity, one fourth of the light sources are selected as key lights. With this change, Equation 5.2 becomes:

$$\mu_{j+1} = (L\mu_{\mathrm{a}} - \sum_{i=1}^{L_{\mathrm{k}}+j} x_i)/(L - L_{\mathrm{k}} - j) \tag{5.4}$$

and Equation 5.3:

$$P(x) = \frac{b}{2}\mathcal{N}(\mu_{\mathrm{k}}, \sigma_{\mathrm{k}}) + \frac{1}{L}\sum_{j=1+L_{\mathrm{k}}/L}^{L} \mathcal{N}(\mu_j, \sigma_{\mathrm{a}}) \tag{5.5}$$

Normalizing Light Brightness

It is common for stage lights to have different maximum brightnesses (different lumen outputs) and vary significantly in the area of the stage they illuminate. We account for this variation by performing a normalization step at the end of the Gibbs sampling procedure. In practice, we normalize the values produced by Gibbs sampling by the total brightness of each light, estimated from HDR images captured as described in Section 5.5. This ensures that each light has a similar global effect on the scene. The system computes the *sensitivity*, s_ℓ, of each light ℓ as the average pixel change in stage renderings when varying the light's intensity by 1% (between 50% and 51%). This is a measure of how much a percentage change in a light's intensity impacts the final appearance of the stage. After Gibbs sampling, the intensity of each light source is scaled by:

$$w_\ell = \frac{1/s_\ell}{\sum_{i=1}^{L} 1/s_i}$$

This scaling ensures that the relative intensities of all lights is normalized so that the sampler's choice of normalized intensity values determines light source brightness independently of the peak brightness of the source.

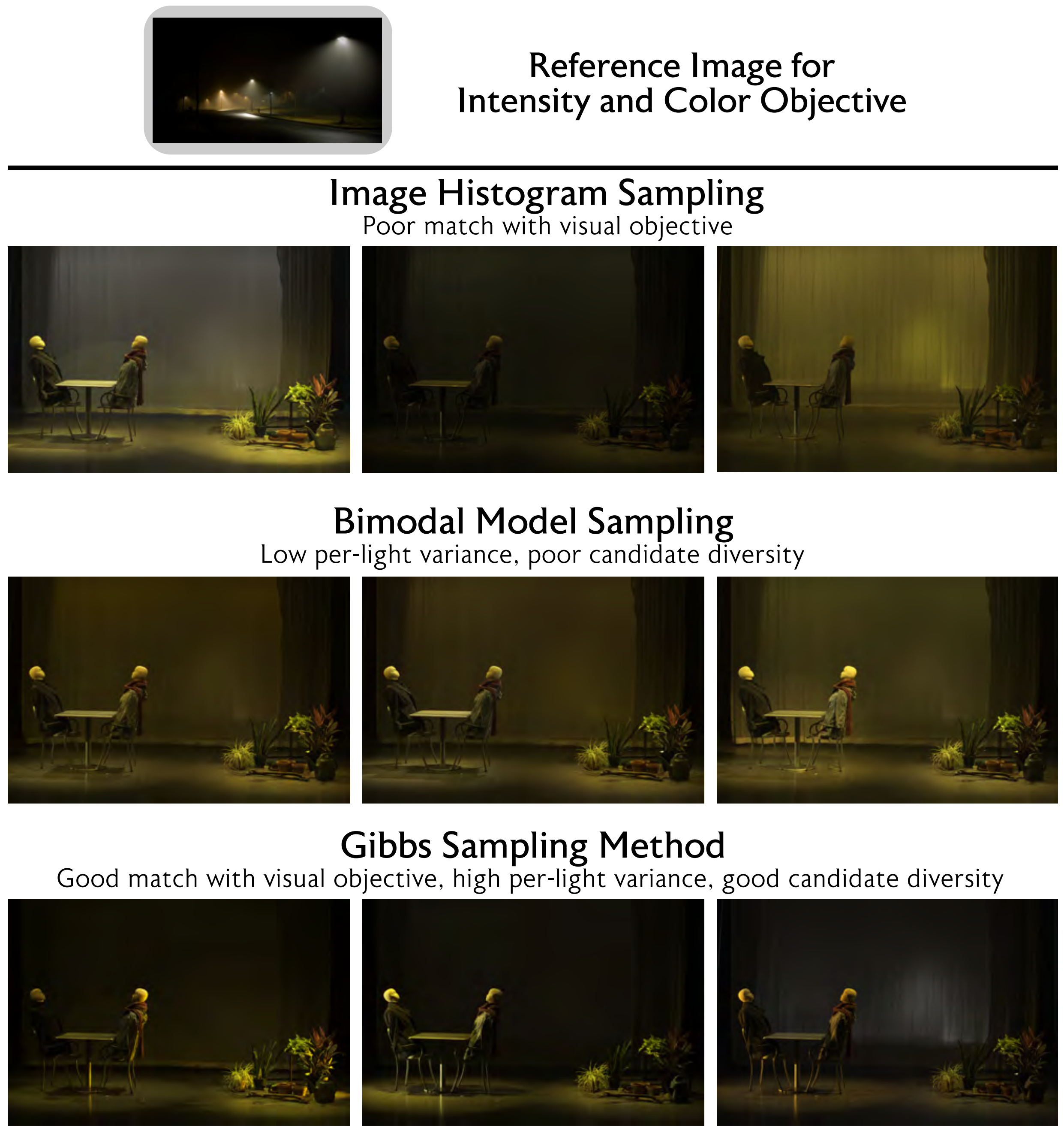

Reference Image for
Intensity and Color Objective
Image Histogram Sampling
Poor match with visual objective
Bimodal Model Sampling
Low per-light variance, poor candidate diversity
Gibbs Sampling Method
Good match with visual objective, high per-light variance, good candidate diversity

Figure 5.4: *Design candidates generated from three sampling techniques.* Sampling light group intensities according to the reference image's pixel intensity histogram yields widely varying results that often do not make the visual appearance of the reference (top). Directly sampling from a bi-modal distribution representing key and fill light intensities (middle) results in candidates that lack the diversity and contrast of those produced by the Gibbs sampling technique (bottom).

Compared to sampling $P(x)$, Gibbs sampling has two useful properties in our context: the average light intensity is more consistent across candidates and individual light sources vary more between candidates. In other words, Gibbs sampling generates candidates that are more consistent with the visual objective and more diverse, both of which are desirable properties for our application. The former property comes from the sequential adaptation of the sampled distribution so that the expected average is μ_{a} conditioned on the previous samples' values. Intuitively, if the average of the previous samples is "off-target", we adjust the distribution so that the expected average including the next sample is back "on-target", thereby reducing the variance of the result. Formally, since $P(x)$ is a sum of independent random variables, the variance of the output is σ_{a}^2/M whereas because Gibbs sampling adjusts the distribution depending on the previous samples, the variance comes from the last sample only, i.e., it is $\sigma_{\mathrm{a}}^2/M^2$ which is an order of magnitude smaller than that of sampling $P(x)$. We now discuss the second property about the variance of individual lights across candidates. For simplicity's sake, we first ignore the key light sources. The variance of a light source ℓ across candidates is σ_{a}^2 for direct sampling and $\sigma_{\mathrm{a}}^2 + \mathrm{Var}_j[\mu_{r_j(\ell)}]$ for Gibbs sampling, where $\mathrm{Var}_j[\cdot]$ is the variance operator across candidates, $r_j(\ell)$ the random index assigned to the light ℓ when generating the candidate j, and $\mu_{r_j(\ell)}$ is computed with Equation 5.2. Since variances are non-negative, this guarantees that the Gibbs sampling variance is larger than that of sampling $P(x)$ directly, i.e., it generates light intensities that vary more from a candidate to another. Adding the key light sources back does not change this property since they are sampled similarly for both sampling procedures. Figure 5.4 illustrates these two properties.

5.4.3 Color Visual Objective

We now describe the color visual objective and how it is used to set light colors during sampling. Given a reference image, the system extracts a palette of C RGB colors, as well as per-color weights. The images are typically JPGs and PNGs encoded in the standard sRGB color space, and clustering occurs within this space. Color weights sum to 1, so they represent the fraction of lights that should take on each color in the palette. To obtain a color palette from an image, our implementation performs K-means clustering on RGB image pixels, then extracts cluster means ($K = C$). The fraction of image pixels belonging to each cluster determines cluster weights. Use of more advanced palette extraction techniques, such as that of Lin et al. [LRFH13] is possible, but their runtimes (minutes per image) are not suitable for interactive use and would require preprocessing of reference images. Users also have the option to manually tweak colors if they are unsatisfied with palettes automatically extracted by the system.

During design candidate generation, color assignment occurs after intensity sampling is complete. For each color c, the sampler uniformly selects a light source whose color has not been assigned, and sets its color to c. The process of assigning c to lights continues until the total normalized intensity of all lights with color c has reached the color's weight. Then the process proceeds to the next color in the palette until a color has been assigned to all lights.

Given the algorithm above, all colors but the last are over-represented in the lighting configuration, so the system randomizes color order for each design candidate. This ensures that the under-represented color varies for all candidates, increasing candidate diversity. For real-world control, we rely on existing lighting control consoles to appropriately handle the conversion from sRGB colors to the proper colors for each lighting device.

5.4.4 Refining the Candidate Generation

While the proposed system is fundamentally an exploratory design tool, we quickly observed that designers benefited from increased control over the visual appearance of design candidates. In this section, we discuss several extensions to the Gibbs sampling process that enhance designer control.

Targeting Regions of the Stage

Designers not only wish to explore different distributions of light colors and intensities, but also different placements of these visual attributes on stage. Therefore, the system allows designers to target a visual objective to a specific region of the stage simply by drawing bounding boxes on the visualizer output (see boxes in Figure 5.2). Given this 2D screen region, the system automatically infers the light sources that affect the selected region, and design candidate generation is constrained to only manipulate the intensity and color values for selected lights.

Light selection from a 2D screen region is performed using a collection of heuristics computed on rendered pixel values. For each light ℓ, we classify rendered pixels as ℓ's *bright pixels* (15% of pixels that receive the most light from ℓ) and *highlight pixels* (top 5%). We set these thresholds empirically and use the same values for all experiments. While these values may not be optimal, our user studies indicated that the system performed as expected in the majority of cases with these values. A light source is added to the selection if:

- The light affects a significant part of the selected area. ($> 25\%$ of the selection is covered by bright pixels)
- The light's influence is mostly contained within the selected area. ($> 50\%$ of the light's bright pixels are contained by the selection)
- The selected area contains highlights caused by the light. ($> 5\%$ of the light's highlight pixels are contained by the selection)

Our approach to inverse light selection is similar in spirit to that of EnvyLight [Pel10], however we select light sources based on 2D screen regions, rather than use 3D surface manipulation to select environment-map pixels. Our experiences indicate that 2D screen-space selection is sufficient to quickly select foreground or background regions of the stage, even without a 3D model of the scene.

Respecting Light Groups

Designers organize lights into light groups to reflect desired spatial and angular coherence of lighting on stage. Although the sampling procedures Sections 5.4.2 and 5.4.3 were described terms of individual light sources, it is most common to apply the sampling procedure at the granularity of entire light groups where assignments of intensity and color are made per group, not per light. Figure 5.5 illustrates the aesthetic benefits of manipulating light groups. Notice that per-light color and intensity assignment (top) results in spatially incoherent illumination across the stage. When a designer targets

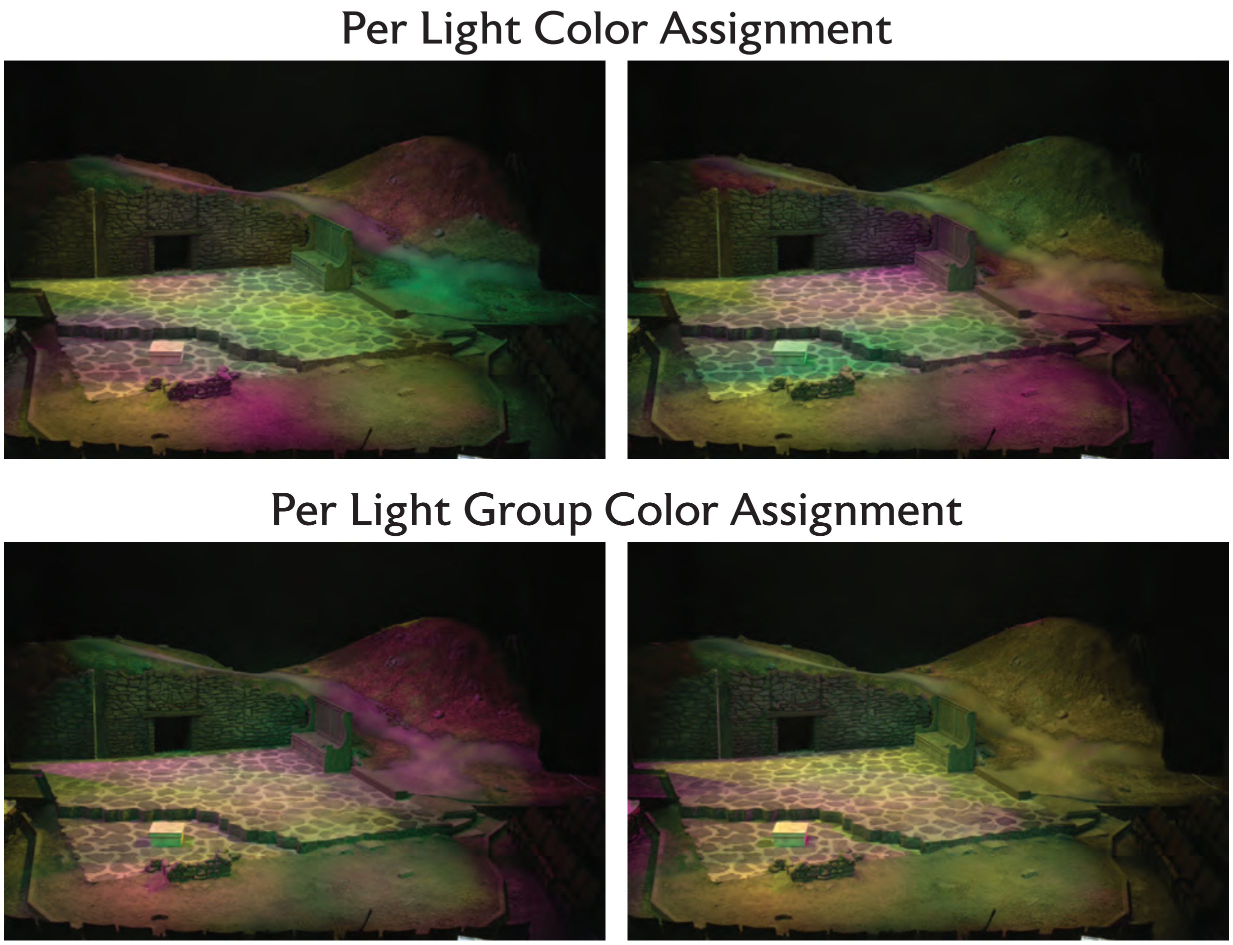


Figure 5.5: Assigning colors at light-group granularity (bottom) yields more pleasing designs (note higher spatial coherence) than when unique colors are assigned to individual lights (top). In both cases design candidates are generated from the same color visual objectives.

visual objectives to a stage region, the sampling process is run over the selected subsets of the light groups.

Partial Design Specification

We have found that the Gibbs sampling based approach easily extends to accommodate scenarios where a designer has a clear view of certain elements of a final design. For example, a designer might choose a particular light group serve as key lights of the scene. This is handled by setting the intensity of these lights first as key lights, and sampling the remaining lights as normal. Because the sampling method is designed to condition future intensity distributions on lighting parameter values that have been previously set, other intensity constraints, such as fixing lights to maintain a given intensity, or lights whose intensity is defined in terms of that of other lights, are also supported by the system.

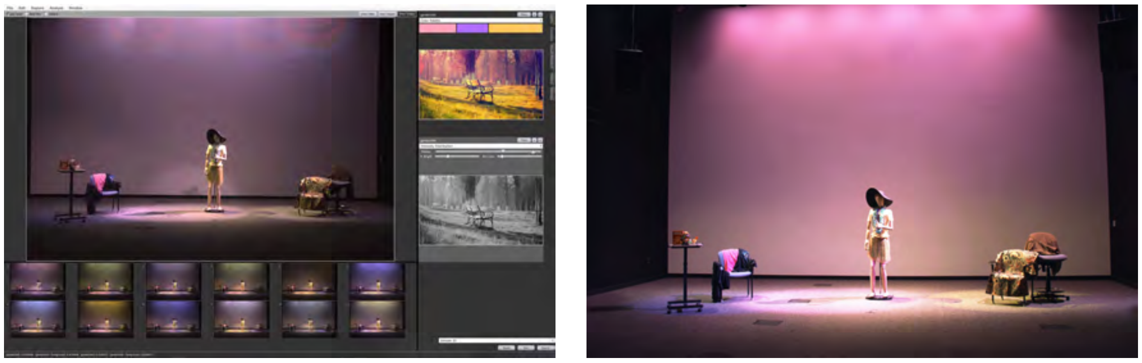


Figure 5.6: Left: Design interface featuring visual objectives and a visualization generated by compositing single-light basis photographs of a stage. Right: a real-life photograph of the same lighting configuration realized on the same stage. In addition to visualizing lighting configurations, the system is also able to directly control the stage lights during the exploratory design process for real-world preview. Note that the right image is missing a spotlight visible in the interface rendering. This is due to a light malfunction that occurred between capture and real-world use.

5.4.5 Presenting Design Candidates

Given a set of visual objectives, the system uses the algorithms described previously to generate a set of N design candidates. The candidates are grouped into c clusters and presented to the designer as a design gallery ($N = 100$ and $c = 12$ in our system). To ensure instantaneous feedback, the system clusters design candidates in a streaming manner. As each new design candidate is generated, the system renders a visualization of the stage corresponding to the new candidate, and measures the average per-pixel L_2 distance (CIELAB color space) to that of existing design candidates. If the distance to all existing design candidates is sufficiently large, a new cluster containing the candidate is made (until c clusters exist). If the distance to an existing candidate is sufficiently small, the new candidate is rejected and the threshold decreased for all future samples. The decreasing threshold encourages diverse clusters to be found quickly. Once c clusters have been generated, all new candidates that are sufficiently diverse are added to existing clusters.

5.5 Implementation

In compliance with the requirements for parameterized design systems (Section 3.2), the visual objectives interface must provide a renderer capable of displaying designs at interactive rates. While this project does not fully implement the design adjectives framework, providing a compatible renderer makes it possible to adopt the framework in future implementations fo visual objectives.

5.5.1 Rendering

Our system provides interactive, photorealistic 2D previews using a custom renderer that is based on HDR image composition. The system accepts as input L HDR basis images that each depict the stage illuminated by a single light at full intensity (all images are from the same viewpoint). Similar to prior systems for cinematic relighting [PVL+05], our tool generates visualizations of different lighting configurations using linear combination of these images. The blend weights for each image are determined by the color and intensity of the corresponding light source. A linear tone map, without gamma correction, is applied at the end of the rendering process.

Compositing single-light images into final lighting visualizations is efficient, even for high light count scenes. We perform simple linear tone mapping to display the resulting image. On a quad-core CPU, the renderer generates hundreds of 480×245 thumbnails per second for a scene with 44 lights (more than could feasibly be displayed at once in an on-screen gallery).

Image composition-based visualization has the added benefit that the visualization is agnostic to the source of the HDR basis images. Basis images may be photographs of the actual target stage, a demo scene environment, or high-quality off-line renderings of a virtual stage environment (if physical access to a stage is not possible, or if lights are not positioned prior to the start of lighting design).

5.5.2 Image Capture

The stage visualizations we display in this chapter were created from basis photographs acquired from real-world theater stages. The capture process is automated with a script that turns one light on to full intensity, captures an image stack for HDR image creation, then repeats the process for all lights. We capture RAW images of each light at different exposures (from -3 to +3 EV in 1 EV increments). HDR images saved in the OpenEXR format [Aca20] were created using Adobe Photoshop CC. After HDR processing, light group information was provided by the designer. The capture process took about 20 seconds per light, and HDR processing using 16 megapixel images took approximately 3 minutes per light on a laptop. More powerful computing hardware and higher quality cameras could notably reduce the length of the stage capture process.

We have captured basis images for four full-scale stage scenes. The first is a 44-light theatrical lighting design laboratory maintained by our university (present in all figures unless otherwise noted). The laboratory is used for lighting design courses and by designers to prototype designs prior to deploying them on a large production stage. Mannequins and props are placed in the stage environment to aid with the design process, which is a standard practice in prototyping lab environments. We captured this stage in four different scenic configurations, each time with the same light sources. The second stage is a 190-light stage used for feature theatrical productions (Figure 5.5). The third stage is a 37-light theatrical lighting design lab (Figure 5.6) that features all color changing fixtures (LEDs, scrollers, and other color mixing systems). The fourth stage (Figure 5.8) is a different configuration of the stage shown in Figure 5.5 with 267 lights. We do not capture multiple scenery configurations of the production stages due to time, however since light positions are fixed, a single set is sufficient for most designers to work with the visualizer. If multiple scenes have been captured of the same lighting rig, the visualization can be swapped out with little cost.

In addition to full size stages, we have also used our system to capture basis images for a miniature (1 m^2) stage mockup (Figure 5.7), provided by a scenic designer to visualize a future stage design before building full-scale scenic elements (see Section 5.6.2). Despite its small scale, we captured

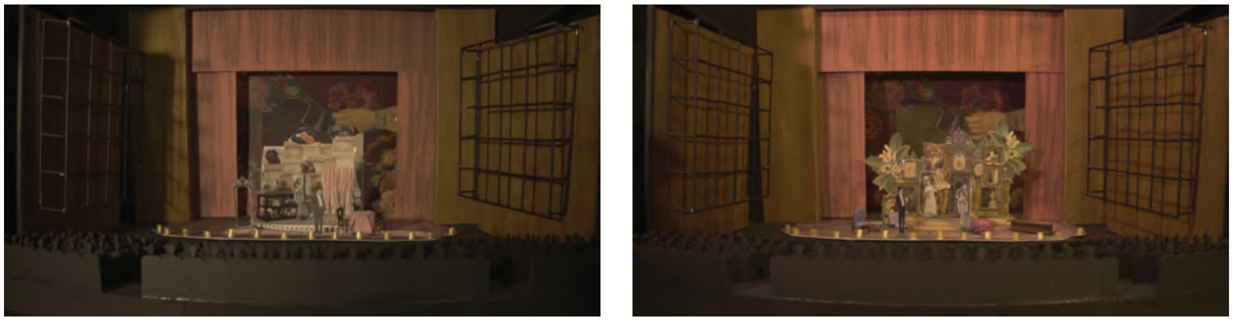


Figure 5.7: Two previz lighting configurations for *The Matchmaker*, visualized on a small-scale (1 m^2) model created by a set designer prior to full-scale sets being built. The model is illuminated by eight lighting directions chosen by the lighting designer.

basis images for the mockup using the same photo capture method as for the full-scale stages, with the position and direction of the lights chosen by the show's designer. We have also used the interface to perform design exercises on virtual lighting stages. In this scenario, we rendered HDR basis images using a photorealistic ray tracer.

5.5.3 Real-World Lighting Control

The speed at which our system generates design candidates makes it possible to use the interface to directly control the lighting configurations on a real stage. In collaboration with Electronic Theatre Controls (ETC), a major lighting control hardware company, we integrated our system with their ETC Eos lighting control system [ETC20]. This integration allows our system to directly control actual stage lights in real time in response to user interaction (Figure 5.6). Integration with the Eos control system provides a comprehensive interface that supports multiple phases of lighting design, from early design explorations, enabled by the interface described in this chapter, to subsequent fine-tuning with industry standard low-level controls. We invite the readers to view the capabilities of this integration in the companion video[1].

5.6 Evaluation

Evaluating a creative tool is difficult, as the goal of the tool is to enable users to iteratively work towards a satisfactory design of their own choosing. Ultimately, the true test of our design is whether the visual concepts interface is adopted by practitioners as part of daily lighting design workflows. Since it is infeasible to engineer a robust, fully-featured system for broad community use, we instead conducted three studies to evaluate the effectiveness of the interface at the professional and novice levels.

At the professional level, we conducted a case study, giving our interface to a professional designer working on a production at our university's School of Drama, simulating an industry-level process as closely as possible. We also provided the interface to a panel of lighting design and control experts

[1] `https://www.youtube.com/watch?v=M8RQXmJbjx0`

for a period of two weeks. Professionals are able to analyze their own design process more easily than novices can, and we conducted interviews with them to determine how well our interface integrates into their workflows and supports their design process. For novices, we conduct a study with the Creativity Support Index [CL14], providing a quantitative measure of how satisfied the participants were with our interface compared to a baseline interface. These three studies cover all of the intended users of the visual concepts system, and the results demonstrate the utility of the interface among all participants.

5.6.1 Professional Evaluation

Expert Panel Feedback

To evaluate the visual objectives interface, we conducted a series of interviews with lighting experts regarding how the exploratory interface affects their lighting design process.

Experimental Setup We interviewed four lighting industry professionals: the graduate student from Section 5.6.2, a community theater designer and lighting control expert, a professional lighting designer, and a lighting control expert with Broadway production experience. Participants were first interviewed about the current state of lighting design and how designers and directors communicate ideas. Participants had knowledge of how our interface worked, but were not able to use it themselves.

After the interview, participants were given the full interface to experiment with for at least two hours at their discretion, after which they were interviewed a second time. The second interview asked participants to consider how the new interface would change the design process and the communication process between director and designer. Note that the interviews were conducted before the designer from Section 5.6.2 designed *The Matchmaker*.

Interviews were transcribed and coded by the authors following standard qualitative evaluation practices [CS08]. The full list of codes is presented in the supplemental material.

Overall Evaluation Overall, the expert users were excited by the possibilities presented by the interface. All participants indicated that our interface makes it easier to communicate abstract lighting design ideas to the director and the rest of the design team, and that the interface would make it easy to experiment with new ideas and set up base looks very quickly. The experts also validated some of the design goals used to build this system, specifically noting that lack of time for iterating on a design is one of the primary limiting factors for creating the ideal design.

Communicating Abstract Ideas In every interview, experts mentioned that the primary challenge with pitching a lighting design to a director is that it is difficult to picture what the show will look like until the artistic team gets into the theater space. As one expert put it "[A lighting designer's] art is how to translate an image into gel (filter) colors and onto a stage. They use very artistic words like vibrant, and muted, or dark; things that try to convey what they're doing" (Expert #2). Failing to accurately convey the design idea to the design team leads to delays and miscommunications similar to those encountered in Section 5.6.2.

Our interface bridges this communication gap by providing a high quality visualization, and a method to quickly turn images into lighting design ideas. The panel agreed that the lighting designs produced by the system "absolutely" (Expert #3) captured the feel of the reference images. One participant strongly preferred the real-world results, saying that the visualization "[did] not do this interface justice" (Expert #1).

The system's speed allowed designers to "take the visual references and translate those relatively quickly into something" (Expert #1). The quality of the generated designs was also good enough to create to create a "really good starting point" (Expert #2) during the tech rehearsal process, when a designer is the most time constrained.

Accelerating Choices Full-color LED fixtures are becoming increasingly common, and designers are now putting off color decisions to the point where "decisions are being made during the technical rehearsal process[, and] there is not enough time during tech rehearsal to actually make all those decisions" (Expert #1). Before LEDs, designers typically had a constrained set of colors available in their gel book (a swatch book for color filters). With the visual objectives interface, we can replace the gel book with a collection of reference images for their favorite color palettes. This "would allow someone to use those reference images and go straight to the stage with those colors" (Expert #2). Color palettes have the added benefit of automatically generating an entire color scheme for the designer, instead of selecting colors one by one.

Helping Novice Designers Participants in the expert study noted that the proposed interface, though motivated by expert design principles, would also "have a huge benefit for novice users who don't understand how you translate a thought in your head onto stage" (Expert #2). Novice users, including high school and community designers, make up a large portion of the lighting controls market. For these users, the ability to quickly and intuitively create compelling lighting designs with little to no tweaking would be very useful.

5.6.2 The Matchmaker: A Case Study

The visual objectives interface was used by a lighting design graduate student to design and program *The Matchmaker*, a main stage show at our university's School of Drama. This student worked in a professional capacity for several years before returning to graduate school. The designer used the interface for both pre-visualization and programming.

Pre-Visualization The lighting designer took a small-scale stage model created by the scenic designer for their show (Figure 5.7) and used the visual objectives interface to resolve a practical design issue in their production. The designer was having trouble convincing the director to allow him to proceed with his desired lighting design due to different mental pictures of what the design would look like on stage. To resolve this, the designer generated a number of lighting design candidates from his reference images and presented a few of the generated designs to the director. The renderings reduced confusion and made the director "much more comfortable", and also helped the designer to "figure out what I [the designer] like about the research and find new research as well".

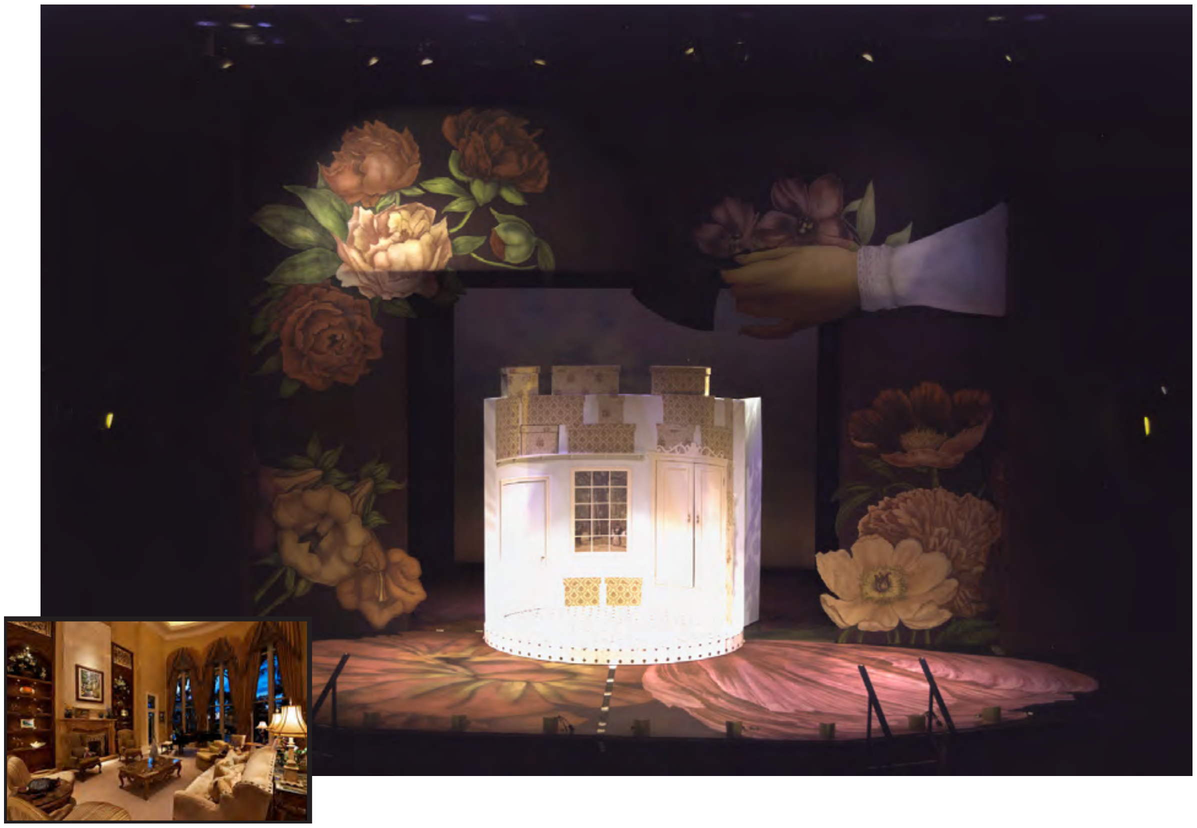


Figure 5.8: *Example Matchmaker Design.* An example of a rendering for *The Matchmaker* and the reference image for the scene. This is the lighting for Act IV shown on the set for Act II (we did not have time to capture every act's scenery). The designs were mainly previewed by controlling lights on the main stage instead of using the visualizer.

Stage Capture The full-size stage for *The Matchmaker* consisted of 267 lights. After the lights were placed on the full stage, we ran an automated capture process to acquire basis HDR imagery in five hours. The interface was integrated with the theater's ETC Eos system, allowing the designer to view the generated designs by controlling lights on the real stage, as well as in the visualizer.

Programming The designer was allocated one work day (eight hours) to program the show according to the production schedule, with the expectation that most of the programming would be complete before on-stage rehearsals began the next day. They were able to explore designs based on their visual research for all four acts of the play within an hour. We only had time to capture scenery of one act, however the designer noted that this was not a problem for him when programming the lights, as the lighting positions were constant between scenes, and he knew where the set would be during the full run of the show. While exploring, the designer used the interface to create presets for each act, which were saved in the Eos console for later use. They then used the remainder of the day to fine-tune the presets with the Eos console (an instance of a sliders-based interface). The show was programmed within the allocated time, which "would not have been possible without [the visual

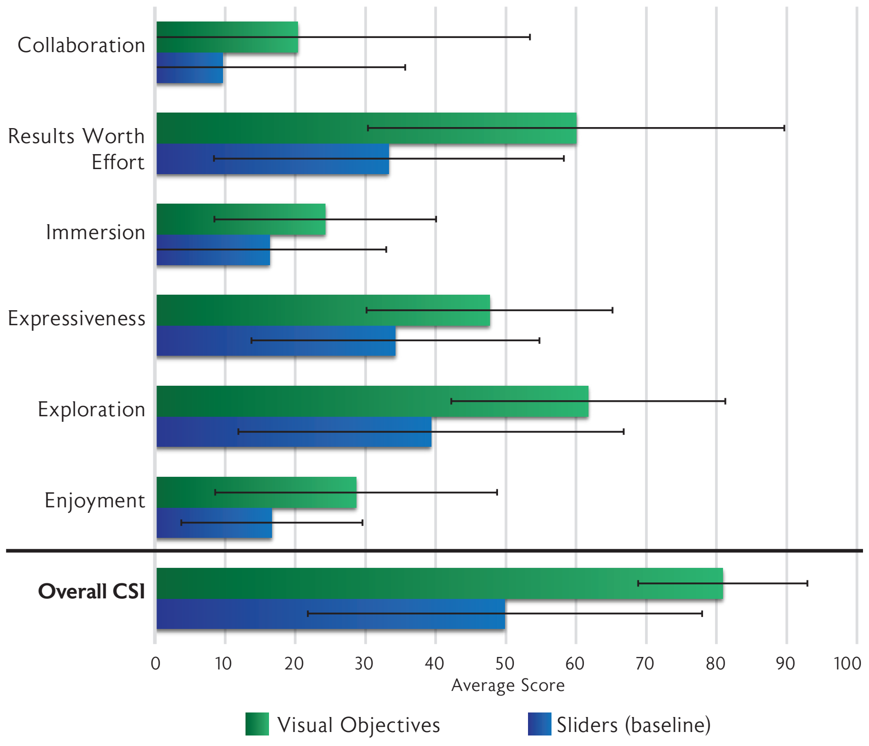


Figure 5.9: *Average CSI Scores.* CSI scores plotted by factor and overall. Error bars represent one standard deviation. The maximum score is 100. The visual objectives interface outperforms the sliders interface at a significance level of $p < 0.05$ in all factors except collaboration.

objectives] interface." An example of a scene created with the visual objectives interface is shown in Figure 5.8.

5.6.3 Novice and Intermediate User Evaluation

We also evaluated the ability of the visual objectives interface to assist novice and intermediate users. Our preliminary studies demonstrated that obtaining an objective measure of lighting design quality is very difficult (Appendix B.1.1), so instead we use the Creativity Support Index (CSI) [CL14], to measure the extent to which our interface supports a user performing a creative lighting design task. The CSI is a weighted average of six self-reported scores measuring the ability of an interface to support enjoyment, exploration, expressiveness, immersion, collaboration, and to minimize effort. Weights are computed from pairwise comparisons indicating which of the six factors were most important to each individual user. CSI scores have a maximum value of 100.

We compared the CSI for the standard sliders-per-light interface with the visual objectives interface added to the sliders-per-light interface. We noticed that expert users liked using the visual objectives interface alongside the sliders instead of completely replacing them, so the visual objectives interface configuration for this study allowed access to the sliders. This setup measured the added benefit of the visual objectives interface to existing interfaces for non-professional use. Both interface configurations used the same renderer, and the task was performed in a virtual environment.

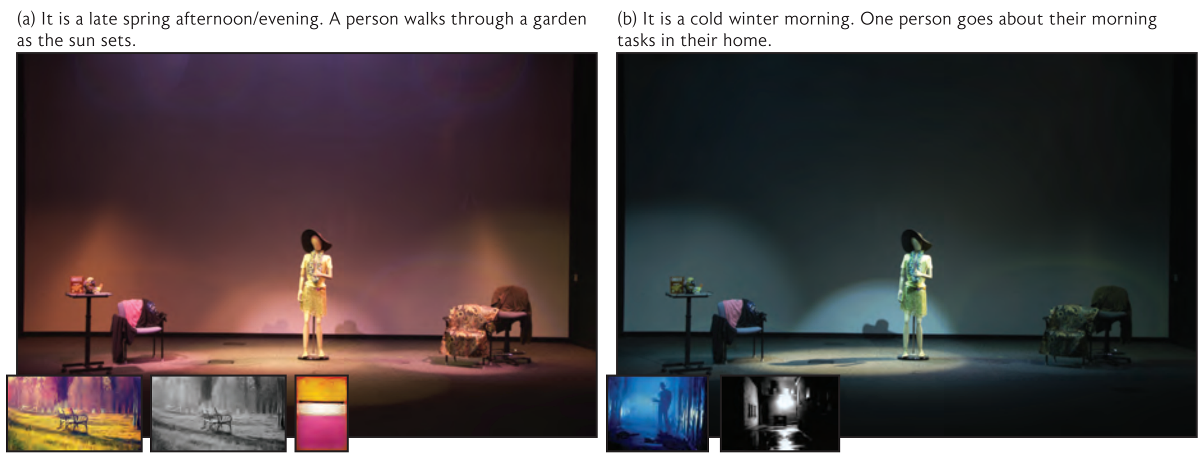

(a) It is a late spring afternoon/evening. A person walks through a garden as the sun sets.
(b) It is a cold winter morning. One person goes about their morning tasks in their home.

Figure 5.10: *Example Novice/Intermediate Study Output.* Scenes created as part of the CSI user study. Both examples were created by participants using the visual concepts interface. The target scene description is shown above the rendering, and the research images chosen by the participant to create each design are displayed on the bottom of the rendering.

Experimental Setup

We recruited 15 participants: 10 with little to no experience with lighting design interfaces, and 5 with moderate experience with lighting design interfaces. 10 participants had prior experience with other visual art.

Each participant was given a 20-30 minute tutorial explaining how to use both interfaces in the study. Following the tutorial, the participant was given a brief description (listed in Figure 5.10) of a scene and instructed to create three lighting designs that match the description. The scene descriptions were taken from an in-class exercise run in our university's introduction to lighting design course. This scenario simulates the situation where a designer must create design variations for a director. Users performed two tasks (one with each interface) and were allowed to take as much time as needed for each task. The CSI survey was administered at the end of each task. The order of the two scene descriptions and the order of the interfaces were randomized.

Since users had limited exposure to lighting design, we pre-selected research images for the visual concepts interface. Participants were allowed to search for and use their own images during the study if they were unsatisfied with our images. Example output for each task is shown in Figure 5.10.

Results

The CSI is a standardized metric that measures creativity support along six factors: Enjoyment, Exploration, Expressiveness, Immersion, Results Worth Effort, and Collaboration. We compute the CSI and the individual CSI factor scores for each interface tested based on the user responses to the standardized CSI survey. The results are summarized in Table 5.1 and Figure 5.9.

The average CSI score for the visual objectives interface was 80.9 ($\sigma = 12.0$) and the average CSI score for the sliders interface was 49.9 ($\sigma = 28.1$) out of a maximum of 100. We note that the visual objectives interface had a higher CSI score than the sliders interface in 14 out of 15 trials. A paired-

CSI Factor	Count (σ)	Paired-Sample T-test
Enjoyment	1.7 (1.2)	$p = .0059$
Exploration	3.7 (1.1)	$p = .000098$
Expressiveness	3.0 (1.2)	$p = .00065$
Immersion	1.7 (1.0)	$p = .008$
Results Worth Effort	3.3 (1.5)	$p = .0011$
Collaboration	1.6 (1.8)	$p = .0571$
Overall CSI		$p = .00012$

Table 5.1: *CSI Counts and T-tests.* CSI factor counts with corresponding t-test. A higher factor count indicates that users valued that dimension more than the others. Counts range from 0 to 5.

sample t-test indicates a significant difference between the two average CSI scores ($p = .00012$). Among individual CSI factors, the visual objectives interface significantly outperforms the sliders interface in Exploration, Expressiveness, and Results Worth Effort ($p << 0.05$). This indicates that the objectives interface betters satisfies our design goals of fast exploration and ease of expression of lighting design ideas compared to the existing control methods. The CSI factor counts indicate which dimensions are important to the users of the interface. We note that the same factors that our system excels in are the most important to the users.

We expect the interface used for each task to be the primary factor affecting the CSI score, but to determine if interface order or scene order also had an effect on the CSI score, we tested the effect of these factors using 3-way ANOVA. The test found no evidence that interface order or scene order ($p >> .05$) had an effect on the CSI score. Additionally, the test found that there is no significant interaction between interface order, scene order, and interface use. The test confirms that the choice of interface configuration is the primary factor that affects the overall CSI score ($p << .05$). Therefore, since the only change in interface configuration was the addition of the visual concepts system on top of the sliders interface, we conclude that the visual concepts system provides significant value to existing systems.

Collaboration This task involves no collaboration, however in accordance with the CSI protocol, we allowed users to mark the collaboration factor as "not applicable" when completing the surveys. Since the task was framed as presenting design options to a director, approximately half of the users chose to answer the collaboration questions in the context of this scenario. Despite answering the questions as applicable, this only influenced the CSI scores for 4 out of the 15 responses, as the collaboration factor count for many users was 0. Our results suggest that the visual objectives interface improves collaboration, however a follow-up study using a collaborative task should be performed to accurately measure this factor.

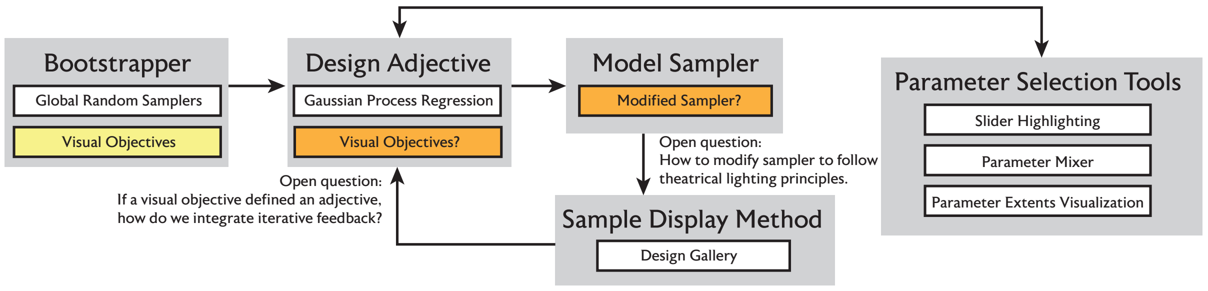

Bootstrapper
Global Random Samplers
Visual Objectives
Design Adjective
Gaussian Process Regression
Visual Objectives?
Model Sampler
Modified Sampler?
Parameter Selection Tools
Slider Highlighting
Parameter Mixer
Parameter Extents Visualization
Open question: How to modify sampler to follow theatrical lighting principles.
Open question: If a visual objective defined an adjective, how do we integrate iterative feedback?
Sample Display Method
Design Gallery

Figure 5.11: *Integrating Visual Objectives into Design Adjectives.* The visual objectives system could be integrated into the design adjectives framework in multiple ways. The visual objectives could simply serve as a bootstrapper for the generic design adjectives (which in turn, would require consideration of how the model sampler will handle design principles), or it could replace the generic Gaussian process regression model of an adjective entirely (which would require consideration of how to handle iterative user feedback in the visual objectives model).

5.7 Discussion

Inspired by the workflows of expert theatrical lighting designers, we have created a new lighting design interface that facilitates the exploratory design process. The key idea of the system is to generate lighting design candidates by mixing and matching visual objectives that abstractly model designer intent. We have evaluated the system in a case study and two user studies, which confirm that the system generates good design suggestions, facilitates communication between lighting designers and the design team, and allows designers to create lighting designs more quickly and easily.

While the positive reaction from designers suggests the current system already has sufficient scope to be a useful design tool in the theatrical context, we are interested in extensions that would allow the system to encompass a broader set of stage lighting design scenarios. For example, while the vast majority of theatrical productions use static lighting, the current system does not model properties of modern moving light fixtures such as adjustable beam position, varying beam textures, atmospheric effects, and the capability to program motion over time. Supporting these new features without compromising system performance or the quality of lighting visualization presents a future challenge. The visualization quality can be further improved with better photometric calibration, tone mapping, and gamma correction methods.

We are encouraged that theatrical designers have been excited to experiment with our interface, and have integrated our interface with ETC's industry standard tools as a proof-of-concept combined interface. We hope our efforts lead to continued collaboration between the entertainment lighting and computer graphics communities. Outside of theatrical lighting design, we believe that similar interface ideas, and our system's core philosophy of using computational techniques to assist a designer explore design possibilities (but not directly solve design problems) may be applicable to other domains such as 3D modeling, photo editing, or other forms of lighting (e.g., image-based lighting for product photography).

5.7.1 Implementing Design Adjectives Components using Visual Objectives

If the design adjectives framework existed when the visual objectives system was being created, it would've been completed in a much shorter time. In order to use visual objectives within the design adjectives framework, we will need to consider which functionality it is intended to replace. Looking at the framework components helps determine how this preliminary system fits in with the overall design process. Figure 5.11 depicts possible components that visual objectives could fit within. If visual objectives were to function as a bootstrapper, it would allow the user to browse through and rate a series of much higher quality designs compared to a randomly generated baseline. However, the model sampler would have to be re-thought, as it would ideally generate new designs consistent with theatrical principles and the domain-agnostic implementation has no knowledge of such principles.

Alternately, the visual objectives model of a design concept could completely replace the generic Gaussian process model of an adjective. In this implementation, the sampling problem is resolved by using the principled sampler from visual objectives, but we are then faced with a question of how to integrate incremental feedback into the visual objectives model. It may be possible in this instance to adjust or learn the visual objectives model parameters based on designs that were highly rated by the user during the design process. These implementations are left as future work.

Chapter 6

Hover Visualizations

The previous two chapters presented systems designed to support the preliminary and refinement design phases, and handled detail design operations with per-parameter controls. Design adjectives used highlighted sliders to help identify relevant per-parameter controls during fine-tuning in a domain-agnostic context. This chapter presents a domain-specific method for identifying parameters called *hover visualizations*, designed for use in layered image editing.

In layered image editing, one of the most common frustrations expressed by artists is the inability to locate layers within the composition [/u/19, art19, /u/11, Wol19, ice19]. The artist's objective is to change a target region's appearance, utilizing one or more layers to accomplish their change. In this domain, it is then natural to ask "what layers affect the region of interest?" While the artist is not changing a design adjective score in this context, this problem can still be thought of as a parameter-finding problem, and the paradigm presented in this chapter can be viewed as a domain-specific parameter editing component in a design adjectives system. Portions of this chapter previously appeared in the paper "Finding Layers Using Hover Visualizations" [SFPF19].

6.1 Layered Image Editing

Layers are a fundamental part of image editing and composition. Layers allow digital artists to separate their composition into discrete parts, and achieve sophisticated effects with varying blend modes and adjustments. Digital artists interact with these layers on a daily basis, editing and arranging them to achieve their artistic vision. Layers in image editing software are typically selected and manipulated through a layers panel (Figure 6.2), which consists of a list of layer names and thumbnails. Some software eschews an explicit layer panel and instead opts to layer elements through simple "move forward/backward" ordering operations. Artist-created compositions have tens to hundreds of layers, many of which are semi-transparent, overlap, or have non-normal blend modes, complicating the seemingly simple task of finding the layer to edit. For example, the composition shown in Figure 6.4-c is an artist-created collage of 125 layers.

In documents such as Figure 6.4-c, it can be difficult to select a layer for editing. Without a layers panel, layers that are occluded by other layers are difficult or impossible to click on. Some software, such as Adobe Photoshop, solves this problem by providing a context menu containing

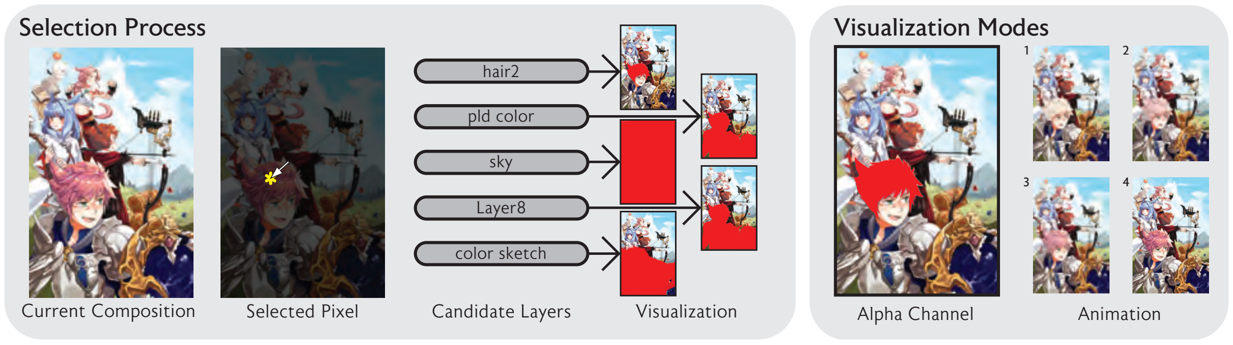


Figure 6.1: *Hover Visualization Interaction Method Overview.* To select a layer, the user first clicks on an area of the current composition. All of the layers that contribute to the selected pixel's color are displayed in a list of candidate layers. Hovering over the candidate layers triggers a visualization on the main canvas. There are two different visualization modes available (right). "Alpha Channel" displays the layer's alpha channel as a solid color. "Animation" displays a brief animation of the layer's opacity, rapidly fading it in and out, shown here as a series of four frames.

all of the layers located under the cursor to allow selecting layers at deeper depths. With a layers panel, there are two ways to locate a layer: by name or through a manual search. Well-named layers provide clear indication of the layers contents (Figure 6.4-a), however many artists do not spend time naming their layers, viewing the process as tedious. This leads to many cases where layers are given non-meaningful names like "Layer 128" and "Layer 8," as seen in Figure 6.4-c Names may also lose their meaning over time, as the artist has the ability to change the content of any layer at will.

The second method of identifying layers in the layers panel is through a manual search. This task is aided by thumbnail previews of the layers. The thumbnails provide limited information due to their size, and their portrayal of the layer pixels in isolation, as they are unable to display the effects of the layer's blend mode without any layers to blend. If the artist is unable to recognize the layer through the thumbnail, they are forced to rely on editing operations to understand what each layer does. The most common operation used is the visibility setting, which they toggle on and off in order to figure out what each layer does [ice19].

While the prior techniques are adequate for compositions with a small number of opaque objects, they are less suitable for finding semi-transparent layers in dense compositions, a common use case for intermediate and expert users. This chapter proposes a set of surprisingly simple UI mechanisms for Photoshop-like applications that enables fast identification of where layers are and what they do. Our design is motivated by observing that in order to quickly identify a target layer, artists need to have the ability to quickly see the effect of their layers in the context of their composition. Our method comprises of a spatial filtering operation to select candidate layers that impact a pixel, and a hover operation on the resulting candidate layer list to reveal the effect of the layer on the composition at large. For this reason, we call our system a "click and hover" interface. We demonstrate the utility of the interface through a user study, finding that in compositions with many overlapping semi-transparent layers, the interface allows layers to be located twice as quickly.

6.2 Layer Selection: Challenges and Design Goals

Layer selection is a universal task in image composition editing. Through our own experiences with these tools, and in conversation with artists, UX designers, illustrators, and students, we believe that frustration with current layer selection interfaces arises from two primary factors. First, the process of finding a layer often requires the modification of the document itself because thumbnails do not provide enough information, and second, that this manual modification search becomes more tedious as the number of layers increases.

Artists will try to use thumbnails to identify a layer, however thumbnails sometimes do not provide enough information to locate a layer. In these cases, the default way to locate layers is to browse through the layer list and toggle the visibility of each layer in order to visualize the changes that it makes to the composition. They often have to repeat this for every layer that they want to locate, and it is difficult for them to remember where it is in the layer panel after they find it. Spatial selection methods help, but these mechanisms do not help when there are many visually-similar overlapping layers at the specified point. Naming layers is viewed as a tedious task by many artists, and even though they know layers should have useful names, they still sometimes choose not to name layers at the time of layer creation. The hover visualization system steps in to help artists find a layer, but labeling that layer properly after locating it is up to the artist.

This problem is exacerbated as the number of layers in the composition grows. With hundreds of layers, it becomes more difficult for an artist to recall what each layer is and what it does in the composition. This problem is further complicated when the layer structure was created by a different artist (i.e. a collaborative workflow) or if the artist has simply not worked with the composition in a long time. One of the most time consuming tasks when working with such documents is figuring out how the document is structured, and then rearranging layers to fit current preferences. This process must be done with the manual modification search process for each layer that the artist needs to locate, which this interface assists with. Artists expressed interest in tools that would allow them to quickly browse through layers in this situation.

Based on conversations with artists, and our own observations, we believe that an interface for fast layer selection should follow the following design goals:

Inline Visualization Artists working in a visual medium need to see how layers affect the composition. They also need to be able to visualize these changes on the composition itself, not in a separate panel or exploded view.

Transient Previews Previews of layer effects should not permanently modify the original composition.

Speed and Scalability Previews should be performed quickly enough to enable rapid, seamless layer browsing as the user hovers over a layer name. The interface should also be able to handle hundreds of layers.

6.3 Prior Work

The hover visualization interface is an example of a parameter selection assistance tool. The interface uses a simple inline visualization to show users what pixels are on a layer before they select it for editing. This is primarily useful for the detail design phase, however use cases may exist where a user selects a few layers for an exploratory design method. While the hover visualization system primarily focuses on image editing, which is traditionally not viewed as a parameterized design space, the approach demonstrated here could be integrated into the prior two projects (highlighting lights for visual objectives, automatically showing parameter extents for design adjectives).

Layer selection in most image editing software [Ado20, The20, SYS20, CEL20] is done through a layer panel (Figure 6.2). This interface displays layers ordered by depth, along with a user-assigned name, thumbnail, and controls for modifying layer depth, visibility, and blending settings. This interface paradigm has remained constant since layers were first added to commercial software packages in 1994 (Figure 6.2-d), ten years after Porter and Duff introduced alpha compositing in 1984 [PD84]. Our method is designed to enhance the current layers panel interface paradigm.

To address the shortcomings of selecting layers in the layer panel, a number of systems have been proposed. Tumbler and Splatter [RRC+06] allows users to access opaque, occluded objects in a composition by applying spatial transformations in order to visualize the depth ordering of the layers in the scene. This interface "explodes" the document in order to show the depth relationships between the layers. The Tumbler and Splatter interaction method applies spatial transformations to the layers, removing them from their original context. With this method, finding layers that have a distinct shape or color among other opaque layers is easy, but finding, for instance, a semi-transparent layer with a non-normal blend mode is difficult, as the transformed layer thumbnail does not represent what the layer looks like in context. This type of interface also has difficulty scaling as the number of layers increases. For instance, Mozilla Firefox 46 [Moz16] had implemented a 3D exploded view to display the structure of web pages, however the it was removed in subsequent versions of Firefox, likely due to scalability issues.

Spatial selection can be used to filter out layers that do not affect a point or region of interest. This type of selection feature has been present in Adobe Photoshop since version 12.0 (2010) [noa12]. Expanding on the concept of spatial selection, LayerFish [WKB+16] presents a touch-based interaction that combines spatial selection with a fisheye menu containing layer thumbnails. The fisheye menu provides thumbnails of the layers with layers that are further apart in depth appearing smaller (creating a fisheye effect). The system appears to be effective for quickly browsing through a large number of layers in a list, however it is unclear how much benefit comes from spatial selection versus the layer thumbnail display mechanism. LayerFish relies on thumbnails and spatial transformations, and while the fisheye menu resolves the problem of working with large numbers of layers seen in [RRC+06], it is still subject to the same difficulties found in performing layer identification with thumbnails alone. Newer systems propose selective undo interfaces [MLL+15], reducing the need for multiple layers in the first place.

Toolglass and Magic Lenses [BSP+93] present a method of previewing changes by utilizing a floating window (lens) which visualizes filters that could be applied to a visual element. These filters can be anything from transparency to complex spatial warps of the image content. The click and

(a)

(b)

(c)

(d)

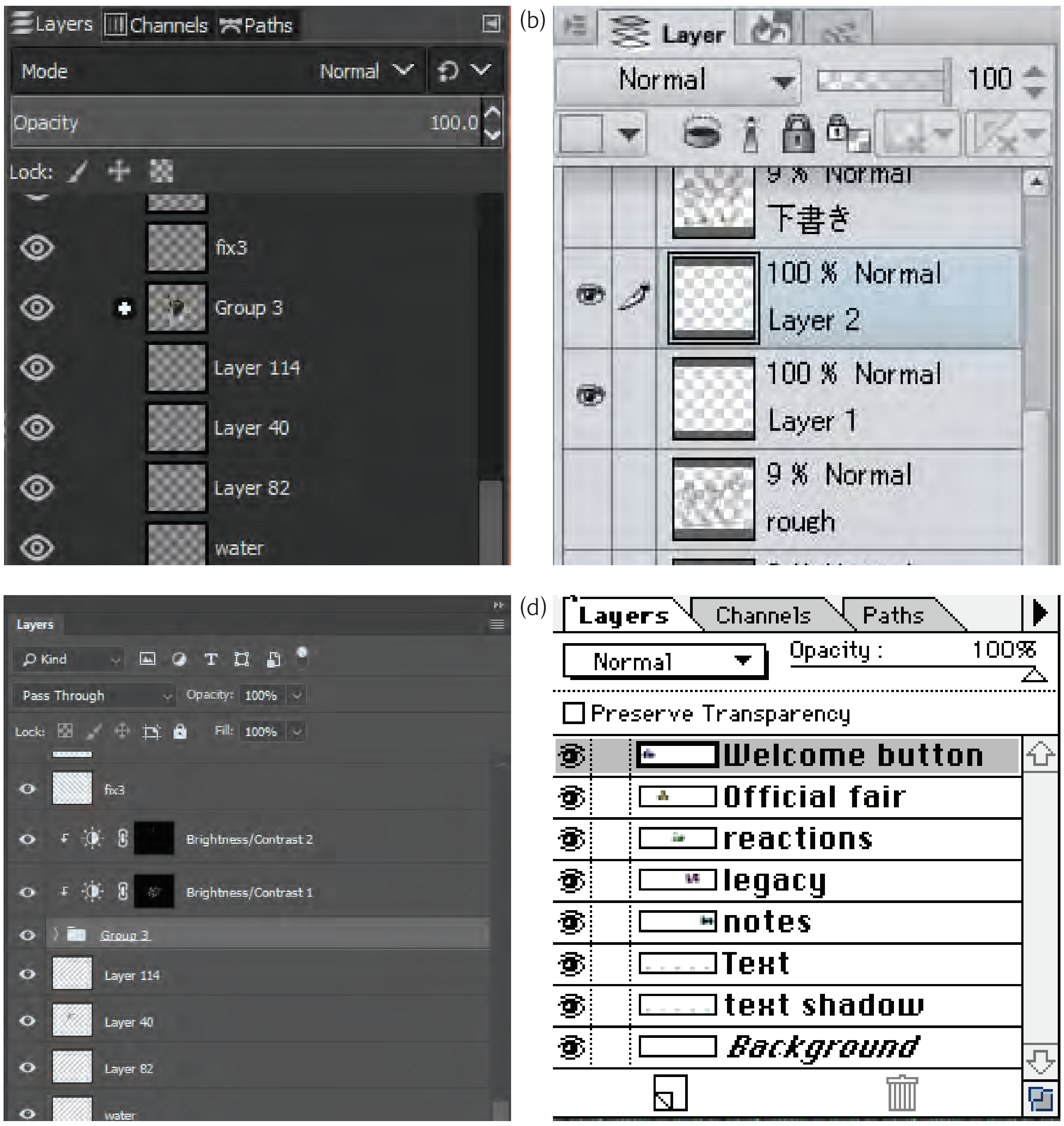


Figure 6.2: *Standard Layer Selection Interface.* Implementations of the standard "Layers" panel in consumer software. Pictured software: GIMP 2.10.6 [The20] (a), Clip Studio Paint 1.7.8 [CEL20] (b), Adobe Photoshop CC 2018 (19.1) [Ado20] (c), and Adobe Photoshop 3.0 (d).

hover system can be viewed as a transient application of a lens that operates on the hovered layer and displays its effect across the entire canvas.

Other methods for assisting layer selection include automatically selecting objects based on selection history [SP]. This method selects groups of objects from a single selected item, using the selection history as a prior that informs how the selection expands. This method is designed for vector objects, and could be extended to bitmap applications, however it does not solve the problem of making the initial selection. Grossman et al. [GBH09] present a selection system for virtual pen and ink systems that combines spatial selection with automatic selection grouping by showing possible selection sets that a single ink stroke could belong to. The click and hover interface can work with these types of automatic selection assistance by providing better tools for making the initial selection.

In windowed environments, multi-blending [BG04] makes content occluded by opaque windows visible by changing the blending mode of the window. Along similar lines, content-aware free-space transparency [IF04] utilizes unused window backgrounds to display occluded content. We are unable to properly use multi-blending techniques in an image editing context, as artists use specific blend modes to achieve specific effects, and we have to respect those artistic choices. In the case of free-space transparency, it is unclear what parts of the artist's composition count as "free-space," and we cannot reliably use this technique because not all of the layers are fully opaque.

In touch-based environments there have been efforts to utilize the "transparent sheets" metaphor of layers to provide intuitive touch controls for layering. Davidson and Han [IF04] present a set of pressure-sensitive touch gestures that support layering tasks on touch interfaces. Hinckley et al. [HYP$^+$10] use observations of how users handle physical pen and paper to inform a set of interaction techniques with a virtual pen and paper. The click and hover interface is not designed to support layer manipulation; instead, the interface supports selection, which is the first step towards manipulating a layer. Supporting fast selection should benefit these types of existing methods by reducing the time spent locating a layer instead of manipulating it.

6.4 Click-and-Hover Interface

To resolve the difficulties associated with finding layers in complex image compositions, we propose a two-step *click and hover* process. We implement this process in our interface, shown in Figure 6.3. Users first perform a spatial filtering operation by *clicking* a point on the canvas where the layer they want exists. Multiple layers may exist at this location, so our interface provides a set of layer visualization tools that allow the user to perform a quick visual search to locate the specific layer they are interested in. The layer visualization tools are activated by *hovering* over the layer name in a separate panel after performing the spatial filter. Once found, our interface allows users to adjust the selected layer through a set of adjustments (e.g. hue, saturation, lightness, contrast, etc.); in a full editor, users would be able to perform edits to the layer pixels. In this section, we describe how the click and hover interface works.

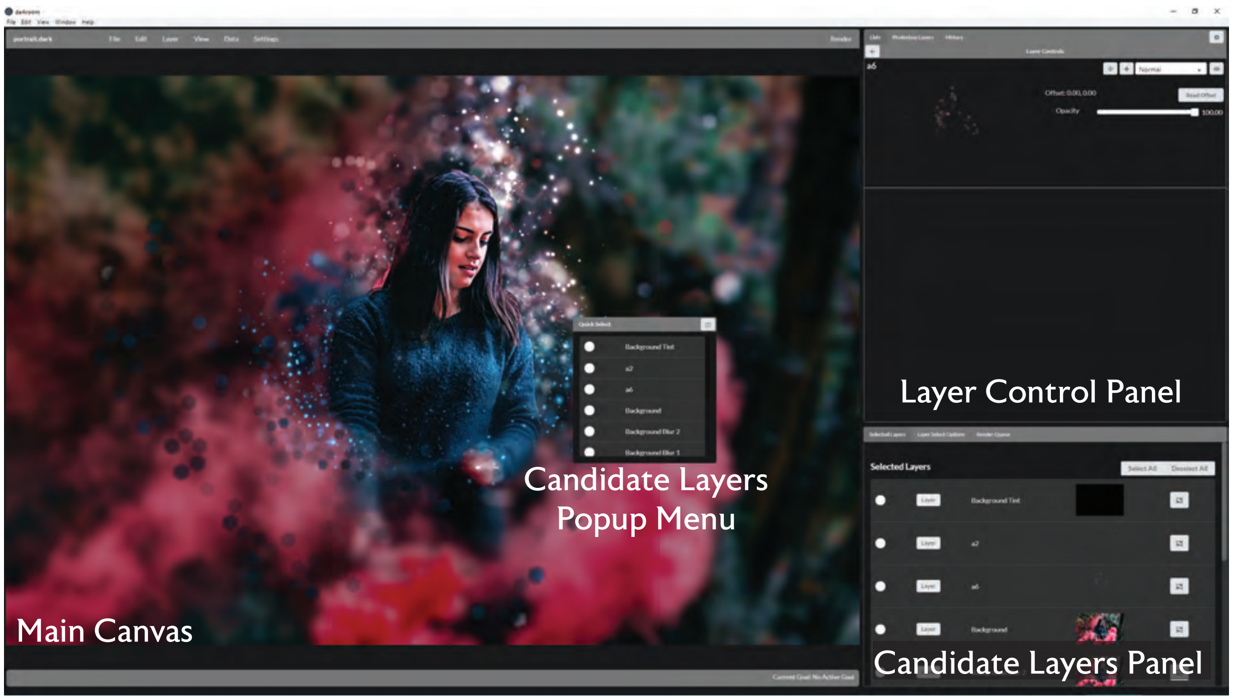


Figure 6.3: *Interface Overview* Our interface is divided into three main sections. The Main Canvas displays the current rendering of the composition. The Candidate Layers Panel displays a list of layers that are under the pixel selected by a user. The Layer Control Panel displays controls for manipulating the layer selected from the candidate layers panel. Users are able to view the candidate layers panel as a pop-up menu by right clicking on the canvas.

6.4.1 Click to Localize

To begin the selection process, the user clicks on a region of the image, and the system computes a set of candidate layers that have a non-zero contribution at the clicked pixel. After determining which layers affect at the selected pixel, the layers are displayed in the *candidate layers panel* in the bottom right side of our interface (Figure 6.3-bottom right). Adobe Photoshop exposes the spatial selection capability through a context menu instead of a panel, so in order to maintain parity with the current state-of-the-art, we also provide the ability to view the candidate layer list in a popup menu (Figure 6.3-center) through a right click operation. The popup menu does not contain thumbnails, but the list of layers shown is identical to the candidate layers panel. This allows users to explore the image spatially, not through the layer hierarchy. This is particularly useful for situations where the user does not know how the layers are ordered, but does know where the elements they would like to modify exist.

6.4.2 Hover to Visualize

In a composited image, multiple layers can contribute to the final color of a pixel in the image. Therefore, in order to easily identify a specific layer, there should to be a mechanism to view the contribution of the layer that affects a pixel in the context of the original document. In the candidate layers panel, we use a hover mechanism that visualizes the hovered layer on the main canvas. Compared to thumbnail views, on-canvas visualizations eliminate the guesswork the artist does to map the transformed thumbnail to a location on the full size canvas, enabling easy visualization of small elements that are hard to see in thumbnails, and providing the opportunity to provide more information about how the layer affects the overall look of the composition. In this way, the user can determine which layers do what with a few waves of the mouse after only one click.

The hover mechanism provides a way to display information about a layer in the same context as the composition. There are many possible ways to visualize the effect of a layer. We propose two such visualization methods inspired by current practices with image editing tools (Figure 6.1-right).

Alpha Channel

One of the primary aspects of a layer that artists look at is shape. The alpha channel visualization displays the alpha channel of the layer, modulated by the current opacity of the layer and the alpha value of the layer pixels, as a solid color on top of the current composition. The resulting solid color layer is alpha blended on top of the current composition. Layers that are more transparent will have a less intense color, and layers that are hidden or do not contribute to the composition will not show up.

This visualization provides instant feedback regarding the shape of the layer on the canvas itself. For occluded objects, the visualization exposes the shape of the occluded part of the layer, as it is presented on top of the final composition. This type of visualization excels when the layers in question have unique outlines and do not cover the entire canvas, but it may lead to some confusion if the hovered layer is occluded, as this visualization will display the layer on top of the overall composition.

The default color for the alpha channel visualization is pure red, the default alpha mask color in Photoshop. This color can be customized by the user if it conflicts with the content of the image composition.

Opacity Animation

The opacity animation visualization makes the hovered layer visible, and then animates its opacity from 0% to 100% over a short duration. The layer's blend mode is unchanged, as is its position in the image composition. This creates a flicker effect that reveals the current effect that the transparency of the layer has on the composition. This visualization method imitates the artist's current layer search methods, specifically that they toggle the visibility of a layer in order to determine what it does.

The animation visualization is designed to provide feedback about how the current layer settings affect the overall composition. This can be helpful for identifying layers that utilize non-normal blend modes, full canvas layers, and layers that are heavily occluded. This visualization does not provide instant feedback, as the animation does take a short time to run.

Our implementation of the opacity animation runs a 25 frame loop at 60 frames per second, animating for 10 frames, and holding the layer at full opacity for 15. This effectively flickers the layer 2.4 times per second. These settings are customizable by the user. In order to run the interface at interactive rates, our implementation of the animation downsamples the image to approximately 25% of the original resolution. Faster renderers should be able to run at higher resolutions.

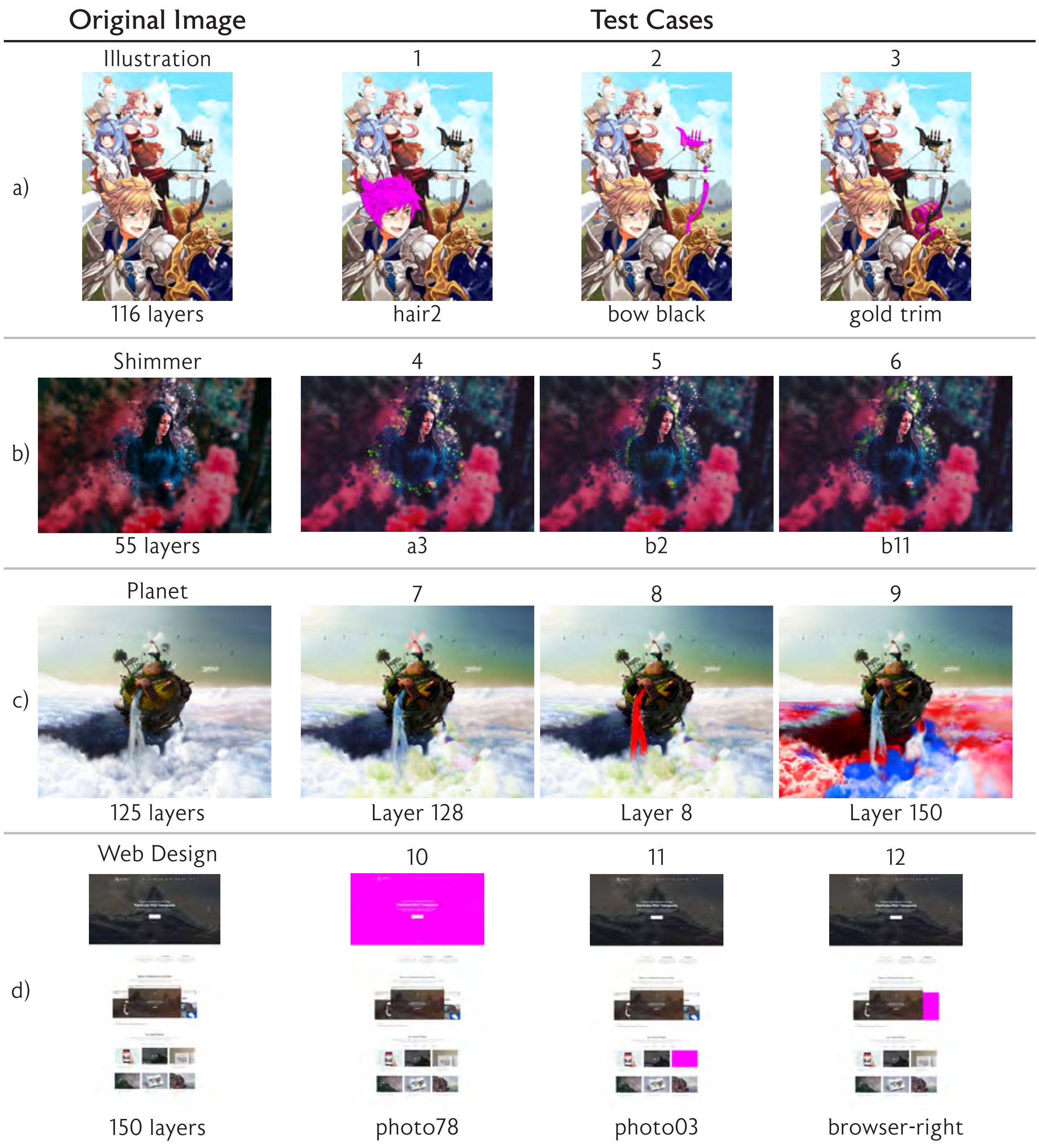


Figure 6.4: *User Study Tasks.* Each task asked the user to find the highlighted layer by performing and overwrite color operation, replacing the color of the layer pixels with a solid color while leaving the alpha channel unchanged. Original compositions are shown on the left with the composition name and total number of non-adjustment layers. The task number is listed above the test cases, and the name of the target layer is listed below the test cases.

6.5 Evaluation

We conducted a within-subjects user study to investigate how much faster the click and hover interface allowed users to locate layers in complex image compositions. Qualitative data about the perception of the effectiveness of the visualization types was also collected.

6.5.1 Methodology

Participants

15 participants age 20-45 were recruited (5 male, 10 female). Participation was voluntary, and no compensation was given. Participants of all skill levels were allowed to participate in the study. Most participants had some experience with software such as Adobe Photoshop, and about 20% of participants described themselves as experts. The participants in this study did not participate in the preliminary interviews described in Section 6.2.

Interface Configuration

Participants used three different configurations of the hover visualization interface: baseline, alpha, and animation.

Baseline The baseline interface configuration allows users to use spatial selection to filter out layers that do not affect the pixel under the cursor. No hover visualizations appeared when hovering over the layer names. This is representative of the current feature set in Photoshop.

Alpha The alpha interface configuration shows the alpha channel visualization described in Section 6.4.2 when a layer name is hovered over in addition to allowing spatial selection. The color of the alpha channel visualization was red, and users were not allowed to change the color.

Animation The animation interface configuration shows the opacity animation visualization described in Section 6.4.2 when a layer name is hovered over in addition to allowing spatial selection. The animation runtime is as described in Section 6.4.2, and users were not allowed to change the timing of the animation.

Participants were not allowed to change the visualization type while using an interface configuration. Participants had the option of using the candidate layers panel or the candidate layers popup menu (Figure 6.3) to display results from the spatial selection. Spatial selection results are ordered top to bottom by layer depth (standard layer panel ordering).

Tasks

In this study, users were asked to find a highlighted layer in an image composition as shown in Figure 6.4. The target images shown in Figure 6.4 are created with an "overwrite color" operation, which changes the color of all the layer pixels to a single RGB color, while leaving the alpha channel alone. Users indicated their layer selection by performing the overwrite color operation with a non-white color (the color did not have to match the highlight color in Figure 6.4). If the user modified

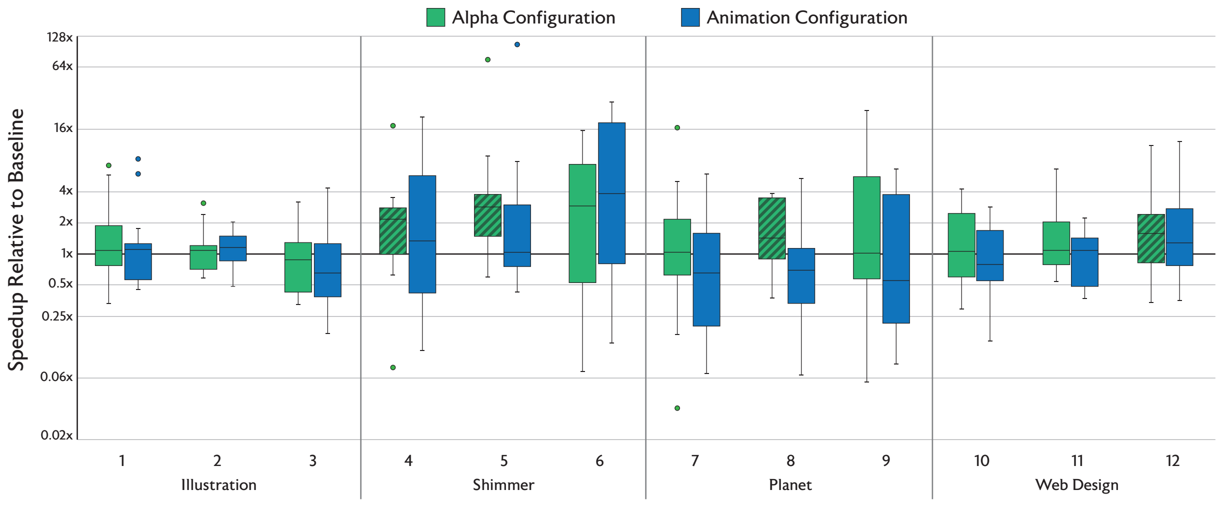


Figure 6.5: *Median Task Speedup Relative to Baseline.* Box and whisker plot of the per-task distributions of the task speedup relative to the baseline. Outliers are shown as single points. The y-axis is plotted on a $\log_2$ scale. The speedup is the time it takes for a user to find the correct layer with the listed interface configuration relative to the time it took for them with the baseline configuration. Results that are significant at the $p < 0.05$ level according to the Wilcoxon rank sum test against the constant baseline distribution (median 1) are indicated with a diagonal striped background. The alpha configuration generally meets or exceeds performance relative to the baseline on all tasks, while the animation configuration appears to help in some circumstances, it does not consistently improve performance.

the correct layer, the task ended. If not, the task continued and the user would select another layer. The task did not end until the user selected the correct layer. Four compositions were chosen to investigate different properties of the hover interface:

Illustration (Figure 6.4-a) 116 Layers. This image consists of well-named layers with sharply defined outlines. Due to appropriate naming and organization of layers, we do not expect the hover visualizations to provide much additional benefit to the users, and performance should be on par with the baseline interface.

Shimmer (Figure 6.4-b) 55 Layers. This image was created by a Photoshop Action, an automated script that applies an effect effect to an input image. The layers created by the script are not semantically named (most consist of a single letter and number, e.g. "a2") and can be difficult to identify from thumbnails alone. This represents a common class of compositions that have a large number of semi-transparent overlapping layers that are difficult to identify with the layers panel alone. We expect the hover visualizations to perform better than the baseline here.

Planet (Figure 6.4-c) 125 Layers. This artist-created image utilizes a number of non-normal blending modes and unnamed layers to create the composition. Some layers appear identical and are layered on top of each other (Task 7), some layers are part of an animated sequence, where only one layer is intended to be visible at a time (Task 8), and some drastically change the appearance of the composition but only when their color is changed (Task 9). These are all difficult cases for the hover visualizations, and we investigate to what extent they help or hinder users in these tasks.

Web Design (Figure 6.4-d) 150 Layers. This composition has the most layers of any composition in our tasks, however, at any particular pixel there are at most four layers. This sort of composition is difficult to navigate with the layers panel alone, but should prove to be easy with spatial selection. As with the Illustration image, we expect this task to be easily completed with all three interface configurations, and include it here to demonstrate the strength of the spatial selection baseline.

Each composition has three tasks, and the tasks are repeated for each interface configuration, leading to a total of 36 tasks. The tasks are performed in sets of 12, where the order of the compositions is randomized, and then the order of the tasks associated with each composition is also randomized. The per-participant task order between interfaces is the same, ensuring that tasks for a particular composition are not inadvertently placed consecutively. The interface order is counterbalanced in order to account for learning effects. Upon completing all 36 tasks, participants answered a short survey. The study took 40-60 minutes to complete.

Metrics

Timing data is collected from interaction logs recorded by the interface. The timer starts when the user is presented with the target layer image. From these logs, we extract timing information including the time to selecting the right layer. We also record the number of times a layer was selected, and the number of identification errors made.

We collect qualitative data from an exit survey. Participants self-reported their experience level with image editing or similar design tools, and answered questions regarding their perception of the hover visualization interfaces. Participants were also given the opportunity to give open feedback regarding what they liked and disliked about the hover visualizations at the end of the survey.

6.5.2 Analysis

Quantitative results from the user study are presented in Figure 6.5 and Table 6.1. Qualitative results are presented in Figure 6.6. Performance on the Shimmer tasks improved by a median factor of 2x with the hover interfaces, performance on the simpler tasks of Illustration and Web Design improved slightly, and performance on the Planet tasks reveal the limits of the presented visualization methods. In particular, we find that the alpha configuration strictly improves on the baseline configuration, demonstrating significantly improved performance on the difficult Shimmer tasks, and performing no worse than the baseline on all other cases. Qualitatively, users enjoyed using the hover interface more than the baseline, praising its ability to provide instant feedback without drawing attention away form the main canvas. The alpha configuration was overwhelmingly preferred over the animation and baseline configurations.

Quantitative Metrics

In order to normalize time metrics between participants, we compute the improvement factor between the baseline and the alpha and animation configurations by using the "time to selecting the correct layer" data recorded in the study. The medians of the improvement factors for each task are shown in Figure 6.5. We report median speedup instead of average speedup due to extreme outliers in some of the tasks. Median and average task completion time, along with standard deviations and total errors, are reported in Table 6.1 to give a sense of the time scale for the tasks.

Where Click and Hover Helps Median performance in tasks involving the Shimmer composition with the alpha configuration improved by 2.1-2.8x ($p < 0.05$ for tasks 4 and 5, $p < 0.11$ for task 6). These tasks involved layers with a large number of separate scattered elements on essentially unnamed layers. Under these circumstances, the alpha visualization let users quickly navigate through the layer stack, and provided more information about the layers location in the canvas than the available thumbnails. In contrast, the animation configuration saw some improvement, but not to the same significance of the alpha configuration.

In these tasks, the hover visualizations also made participants more precise, reducing the number of errors made from a total of 30 in the baseline, to 7 and 6 for the alpha and animation configurations respectively. Errors are defined as the number of times the wrong layers were modified with an overwrite color operation before modifying the correct layer. In other words, the participants did not need to modify the composition as much with the alpha and animation configurations as they did with the baseline configuration, satisfying the transient previews design goal of the click and hover interface. Hover visualizations also reduced the wall clock completion time variance in tasks 4-6, suggesting that all participants had an easier time completing the task with visualization configurations. Most participants favored the alpha channel visualization, viewing the animation as too slow and not providing enough feedback about the boundaries of the hovered layer, however participants were more precise with both hover visualizations configurations compared to the baseline.

Hover Visualiations Do Not Hinder Simple Cases In cases where the layers are well named or sparse, the hover visualizations do not impede the layer selection process, and should be safe to add to existing image editing software without adversely affecting existing processes. We observe that there is no significant difference in speedups in either the alpha or animation configurations for tasks involving the Illustration and Web Design compositions. Task performance on Illustration is best explained by noting that all of the target layers had semantically meaningful names (i.e. "Hair2" corresponds to a layer containing a character's hair) and used mostly standard blend modes. Under these circumstances, users could simply use the layer names without relying on the hover visualizations.

On the Web Design tasks, spatial selection provided participants with a list of at most three layers when looking for the highlighted layer. The time it takes to manually inspect those layers is negligible, and performance improvements were minor. We note that task 12 had a median improvement of 1.5x when using the alpha configuration, which was significant ($p < 0.05$). This task involved finding an occluded layer, and some users found the thumbnail to be unhelpful in this circumstance.

		Baseline			Alpha			Animation		
Composition	Task	Median	Avg (σ)	Err	Median	Avg (σ)	Err	Median	Avg (σ)	Err
Illustration	1	5.4	9.7 (10.0)	1	**5.0**	**5.0 (3.4)**	**0**	5.5	6.7 (3.3)	**0**
	2	6.4	7.2 (3.2)	0	6.0	**6.7 (2.9)**	0	**5.6**	6.8 (4.3)	0
	3	**7.2**	**7.9 (4.1)**	**0**	7.5	10.6 (7.3)	**0**	11.1	12.1 (7.6)	1
Shimmer	4	34.9	57.0 (57.4)	9	20.5	**37.0 (54.7)**	4	**19.6**	37.2 (40.3)	**0**
	5	35.4	107.3 (169.1)	9	**12.6**	**23.3 (20.7)**	**2**	29.5	38.7 (25.2)	5
	6	54.6	103.0 (113.3)	12	18.0	40.5 (53.7)	1	**14.9**	**26.7 (29.1)**	**1**
Planet	7	20.6	30.5 (32.9)	**4**	**13.8**	**28.5 (28.5)**	9	27.9	44.5 (39.8)	11
	8	32.7	29.8 (18.3)	1	**12.5**	**16.6 (8.9)**	**0**	34.1	46.7 (35.0)	2
	9	43.0	56.1 (61.2)	8	**28.8**	**33.3 (29.1)**	**6**	47.6	66.9 (60.2)	9
Web Design	10	5.4	6.5 (3.6)	**0**	**5.0**	**5.7 (3.9)**	**0**	6.2	7.7 (7.2)	1
	11	5.6	5.7 (2.7)	0	**4.2**	**4.5 (2.0)**	0	5.8	6.3 (2.7)	0
	12	8.3	16.1 (21.3)	3	**5.5**	**6.4 (2.5)**	**0**	7.2	7.6 (3.9)	**0**

Table 6.1: *Wall Clock Task Times and Total Errors.* Median and average completion times (with σ) for each task in seconds. Lower is better. Errors are defined as the number of times a participant incorrectly identified the layer in the task before identifying the correct layer, and the total number of errors made by participants on each task is reported here. Best values for each metric in each task are bolded.

Where Hover Visualizations Struggle Median relative performance in the tasks involving the Planet composition generally went down while using the animation configuration (except in task 8), and were approximately equivalent to baseline while using the alpha configuration. Each task in this composition has some unique challenges highlighting the limits of the visualizations presented here.

In task 7, there are two identical layers located at the highlighted location ("Layer 128" and "Layer 128 Copy"). Both layers were depth-adjacent and appeared next to each other in the candidate layers list. Layer 128 is above Layer 128 Copy and was placed before Layer 128 Copy in the candidate layers list. Neither visualization was quite able to help participants identify which of the two similar layers is highlighted in the target image, as the only difference is a slight transparency differences. This situation had fewer errors using the baseline configuration, likely because participants worked from the top of the candidate layer list to the bottom, instead of immediately exploring the other options as they did with the hover visualizations.

Task 8's target layer is co-located with a large number of similar layers. There is only one layer (the target, Layer 8) that is currently visible, all other similar layers are turned off (opacity 0%). In this instance, the alpha visualization will only activate on the one visible layer, making this particular task almost trivial and providing a median 1.4x speedup. The animation visualization did not provide any assistance on this task, as it does not respect the current visibility settings of the layers. Because the animation provided no assistance, the task in this configuration was reduced to the baseline.

Task 9 asked participants to identify a layer using a non-standard blend mode. The specifics of this task caused problems for the visualizations due to the layer only revealing its color effects when a non-white color adjustment is applied. In this case, the animation configuration failed to provide useful information by just adjusting the transparency, and the alpha configuration had trouble because it was difficult for participants to tell where the highlighted layer applied the relevant effect.

Learning Effects There may be a learning effect within participants, as they may be able to remember the layer names for the compositions in subsequent trials. To evaluate the strength of this effect, we ran an 1-way ANOVA over the per-task ratio distributions for the alpha and animation configurations, using interface order as the grouping variable. If there is a strong learning effect, we would expect the means of the ratios grouped by order to be different. We found no significant effect from order on the alpha configuration ($p < 0.05$), but did find a a significant effect on tasks 6, 8, and 10 ($p < 0.05$) with the animation configuration.

The learning effect on task 8 is likely due to the animation visualization providing no additional information for that task. In later repetitions of the tasks, participants generally ignored the animation visualization and repeated the process they had already done in the baseline configuration, leading to faster task completion times. The learning effect on task 6 is possibly due to the difficulty in seeing the layer's effect, as it is towards the bottom of the layer hierarchy. Users may have remembered the difficulty of this layer, and relied on that instead of the animation visualization. Task 10's learning effect is difficult to explain. The interface choice did not significantly affect the performance on this task, and yet there is a statistically significant ordering effect.

Qualitative Metrics

Responses to the survey questions are presented in Figure 6.6. Overall, participants strongly preferred using the hover configurations over the baseline interface, stating that it was both easier and faster to find layers using the alpha channel visualization. One participant explained their responses by saying that "the Alpha channel required very little cognitive effort. [The visualizations] would be very useful on compositions with a lot of layers with intricate effects." (participant 5)

The color of the alpha channel visualization was fixed to pure red in the alpha configuration. Almost all participants found this color choice to be acceptable for the four compositions in the study, although one participant remarked that they "would like to be able to change the highlight color – some colors are more appropriate to some tasks than others." (participant 9)

The ability to obtain instant feedback without needing to look at different parts of the interface to determine layer location and effect is viewed as a positive by many participants. Participants specifically praised the click and hover system for providing "instant feedback about the layer [without] having to look at the little [thumbnail]" (participant 8) and that the alpha visualization specifically is "is easy to flip through" (participant 9). This is a good indication that the click and hover system accomplishes the speed and scalability design goal presented in Section 6.2.

Participants viewed the alpha channel as faster than the animation visualization, and given a choice, 80% of participants expressed interest in only using the alpha channel visualization. The speed was one of the primary concerns that participants had with the animation. Many noted that the animation felt too slow and wanted it sped up. Speeding up the animation further may create a strobe effect that would be unsuitable for photosensitive users, so any implementation of this type of visualization should be be mindful of that effect.

Participants expressed some interest in putting hover visualizations in existing commercial image editing software. If it were to be implemented, participants noted that they would like the ability to control which visualization get activated, as they found that the visualization could be jarring.

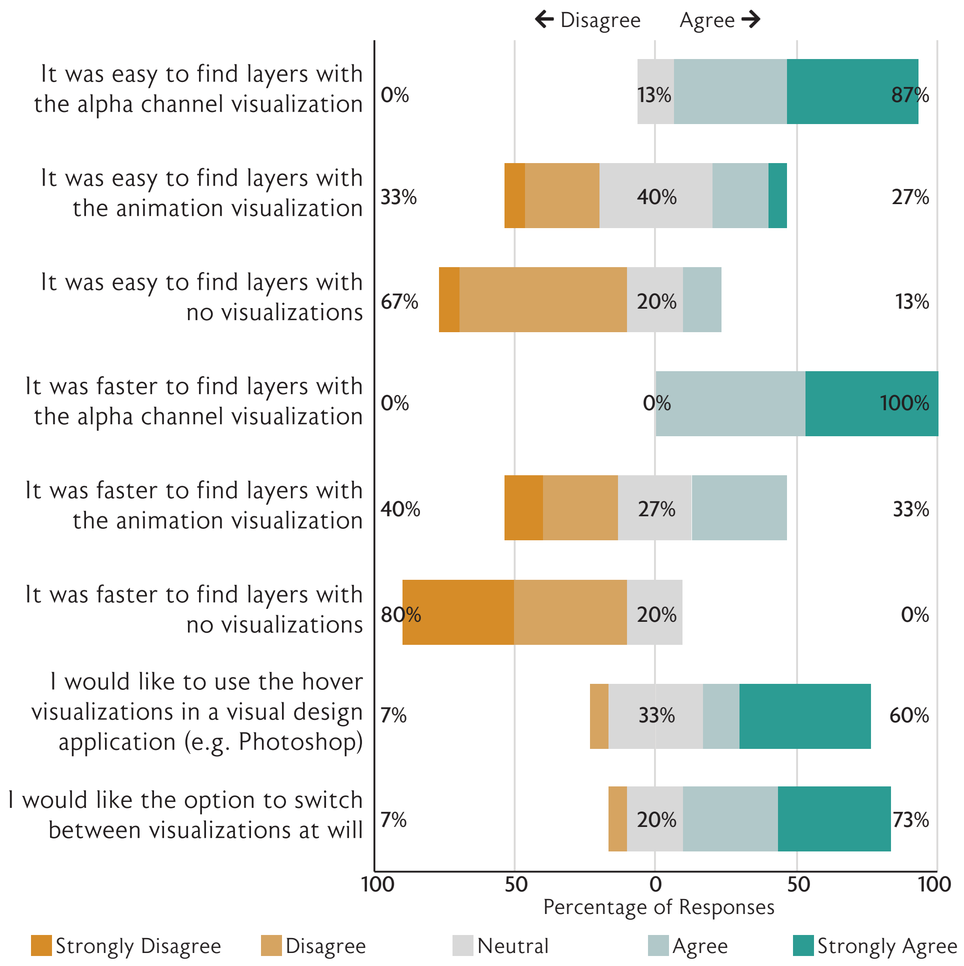


Figure 6.6: *Survey Responses.* Likert plot of responses to the survey questions. Participants overwhelmingly favored the alpha channel visualization, and most expressed interest in bringing the hover visualizations (in some form) to commercial editing software.

We recommend that implementations of the visualizations in image editing software provide the adequate controls to fit visualizations to user preferences.

6.6 Future Work

This chapter presented the click and hover interface and evaluated two visualizations that can be used with the system. Through our evaluation of the visualizations, we find that the click and hover system effectively augments the existing layer manipulation interfaces, providing additional information about the location and effect of layers that names alone cannot provide.

We are interested in continuing to explore new hover visualization techniques. Our current hover visualizations provide insight into where the layers are in the composition, but an interesting area of future work is to explore hover visualization techniques that better suggest what could be done with the layer. The ability to provide speculative visualizations, showing what the user could do instead of what is currently being done, may resolve the difficulties encountered in the our study's Planet tasks. Some additional visualization techniques to consider include:

"Marching Ants" This visualization method places an dashed outline that moves along the edges of a selection. Commonly found in image editing software, this visualization may prove to be less jarring than the alpha visualization, while conveying the same information.

Occluded Alpha Similar to the alpha visualization, except that instead of rendering the alpha channel on top of all layers, the mask is rendered at the same place as the layer in question.

Alpha Mask The alpha channel of the layer is used as a mask to darken areas of the image that are unaffected by the hovered layer

Color Adjustment As noted in the user study, some layers do not reveal what effect they have on the composition. Possible options for this visualization include rotating the hue in an animated effect or inverting the layer instead of using a solid color.

The click and hover interface is designed for traditional mouse and keyboard interfaces, but can be extended to touch interfaces. Instead of a click and hover action, a touch interface could utilize a press and slide interaction technique. In this interface, the layer visualizations would be activated by the user pressing and holding on a location on the canvas (click), and could activate visualizations by sliding up and down to scroll through the candidate layers list (hover). The visualization techniques remain the same, but the interactions that activate them change to fit the properties of touch-based input interfaces.

In addition to layered image editing, hover visualizations can also be used in design domains where the output of a parameter affects a particular region of the rendered image. For example, hover visualizations could be used in theatrical lighting design contexts to highlight what region of the stage one light affects (Chapter 5). For general parameterized design, it is less clear how to integrate the hover visualization system, however the results of this project suggest that rapid parameter identification tools would be helpful for such tasks. The approach taken in the design adjectives system in Chapter 3 was to present a set of thumbnails showing snapshots of individual parameter values along the range of the parameter. A hover visualization version of that display would visually animate the parameters, or display some sort of effect indicating the parameter effect. The exact form of the visualization will depend on the specifics of the design domain.

Chapter 7

Conclusion

The systems built and tested in this thesis demonstrate how fast approximate modelling and sampling techniques can better support exploratory design, and demonstrate how simple per-parameter interface adjustments can better support detail design. The success of the sampling systems suggests that exploratory design interfaces do not need to immediately identify the one “correct” solution to the design problem, but should instead allow the user to quickly determine which regions of the space are likely to contain a satisfactory solution. Thinking about the process of exploratory design as region-finding instead of solution-finding brings up a number of interesting observations, including the idea that a model of a preference function only needs to be accurate enough to find interesting regions of the design space, that priors should be used to bootstrap preliminary sampling and will need to be carefully integrated with design adjectives-like refinement samplers, and that per-parameter selection tools can help support exploratory design as a first-pass filter.

7.1 Modeling Subjective Functions

The results of the design adjectives and visual objectives systems indicate that users prefer tools that rapidly bring them to interesting regions of the design space rather than simply returning a single “correct” result. These systems accomplish this by building models of the user’s design preference function and use a fast sampling method to generate designs based on that model. Back in Chapter 2, I made the claim that capturing user design preference in an objective function was almost impossible. But in Chapter 3, I demonstrated that a rather unsophisticated machine learning algorithm could learn *something* about design intent from a small number of labeled examples in the space, and that intent can be captured with a simple model created from expert knowledge in Chapter 5. Have we then solved that problem of accurately modeling preferences outlined in the prior work? I would not argue that either sampling-based project provides a completely accurate representation of a user’s preferences. However, I would argue that these projects demonstrate that a model can be created that captures a sufficient amount of intent to allow a user to make progress towards a design goal.

This result is not entirely surprising. We know that the design process is iterative and that the detail phase of design is almost always present in the process (Section 2.1.2). From this understanding of the design process, we know that almost any output from any generative system will be tweaked

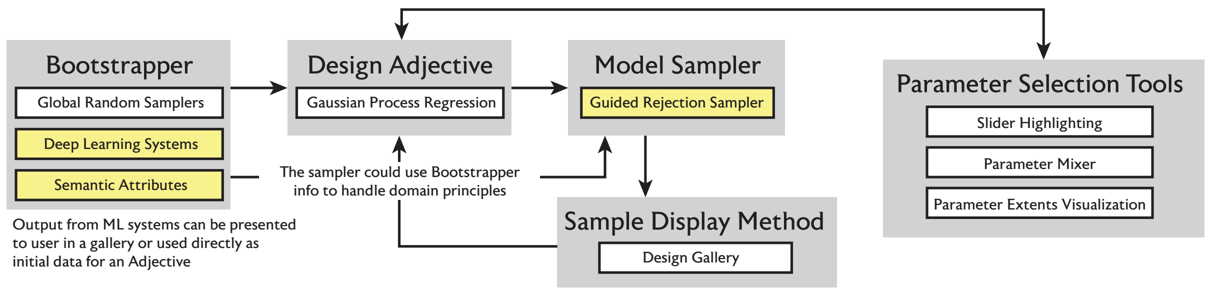

Bootstrapper
Global Random Samplers
Deep Learning Systems
Semantic Attributes
Output from ML systems can be presented to user in a gallery or used directly as initial data for an Adjective
Design Adjective
Gaussian Process Regression
The sampler could use Bootstrapper info to handle domain principles
Model Sampler
Guided Rejection Sampler
Sample Display Method
Design Gallery
Parameter Selection Tools
Slider Highlighting
Parameter Mixer
Parameter Extents Visualization

Figure 7.1: *Integrating Machine Learning Systems with Design Adjectives.* Under the design adjectives framework, other machine learning systems can be used as bootstrappers for the design adjective model. Output from the ML systems must be in the original parameter space. Generated designs can be presented to the user in a gallery, similar to existing bootstrappers, or used directly as a prior for the adjective. It is likely that the model sampler will need to be modified to handle design principles, and access to the bootstrapper model could be provided.

or adjusted by the designer in the detail design phase. As an example, the graphic designer who made the fonts in Section 4.6.2 stated that they always make small tweaks to individual glyphs from any font that they select for detailing, which includes the fonts they created for that project. Getting this level of precision from a model of a subjective design function is then a waste of time. Why bother if the designs are going to be adjusted at the last minute anyway? It appears that it would be more useful to have a system that gets designers *to the point where they can focus on detail design* as quickly as possible, and we were told as much in the evaluations of the tools.

If this is true, one might ask "if everything is going to be changed, why try to estimate preferences for anything?" This would leave us with the per-parameter slider interface, which, as we have seen in all prior chapters, is less preferred for exploratory design. Exploratory design is about finding *regions* of the design space that are viable for detailed exploration. Constraints can be used to find such regions, in domains that have physical restrictions or the set of viable designs can be explicitly specified (i.e. making sure a texture isn't overexposed). Designers want to be the part of the system that makes the value judgement, and the computer should be used to suggest relevant regions of the design space, but should not force a designer into a specific design.

Building systems that represent regions of the design space may provide a way to build human-understandable machine learning systems. Users of the design adjectives system felt that the system understood what their design concept was and understood how to get the model to better understand the concept when it fell short. Allowing the algorithm to be less accurate and trusting the human-in-the-loop to complete the task to their satisfaction may be a promising direction for future creative AI tasks.

7.2 Design Adjectives as a Deep Learning Front-End

Design adjectives could be used as a front-end interface for working with and fine-tuning the output of existing deep learning techniques for assisting creative tasks. In this proposal, the deep learning system would serve as a bootstrapper to the existing design adjectives system (Figure 7.1), providing initial examples according to the criteria defined by the capabilities of the network. As an example, this system could be used to customize the output of semantic attributes. The user would ask the attribute to generate designs according to the learned attribute. These generated designs would be used as positive examples for a newly defined adjective, and users would then be able to customize future generated designs by providing feedback through the standard design adjectives framework methods. This method of adjusting the output of deep networks may make them more palatable for creative tasks, as designers would have means of customizing the output of these black box systems.

Getting this front-end system to work would require some thought about how much weight to put on the initially generated examples. In the current implementation of design adjectives, a single user-provided example would have the same weight as any of the automatically generated examples, which is not desireable for a user-customizable system. Possible solutions to this problem include having the user quickly sort through and score the output of the deep network, thereby making sure the scores are human-assigned, or applying a weighting scheme to the scores such that user-provided examples will always have more impact than automatically generated scores. This method of interfacing with deep networks treats them as a prior to the design adjectives system.

7.3 The Role of Design Space Priors

Priors, in the machine learning sense of a function that provides initial conditions for a model, and design principles can be used on a per-domain basis to improve the quality of an initial set of samples returned during the preliminary design phase. Ideally, better preliminary designs would reduce the number of refinement iterations needed to arrive at the point where only small per-parameter adjustments need to be made. To illustrate how such a system would work, let's consider how we could use ideas from visual objectives as components in the design adjectives framework. The design adjectives system described in Chapter 3 primarily addressed the refinement design phase, and left creating a preliminary set of designs up to random sampling. This approach works in design spaces where most random parameter configurations create valid designs, but is likely to struggle in domains where parameters have fewer valid configurations, such as the stage lighting design domain.

To address this issue within the Gaussian process regression adjective framework, we can add priors to the definition of the adjective. These priors should enforce basic design principles for the user (making sure a texture is properly exposed, the design is physically valid, etc.) but should not attempt to make a subjective value judgement about the "quality" of a design, since that's what the adjective is for. This prior could then be sampled from and used in an adjective to improve the quality of the designs suggested by the adjective. However, the definition of a visual objective does not fit nicely within the structure of a Gaussian process; it is not immediately clear how the sampling method for generating a design would function as a prior. Regardless of how an adjective works, we already know that the output of the visual objectives system is a set of designs that are primed for further refinement. The question then is how to design a refinement sampler that maintains the design principles encoded in the visual objectives model.

One way to solve this problem is to learn preferred parameters of the visual objectives lighting model, and then to generate samples from the preferred model. The generated designs would still be representable in the original design space, but the rejection sampler proposes candidates by using the principled model. Users would be able to specify parameters that should be fixed, a feature already handled by the visual objectives sampler. The drawback of this approach is that the design adjective would not be able to help identify individual parameters that are important, as we are now learning the user's preferences regarding the *model parameters* instead of the low-level parameter values. Another approach could use the design adjectives system as written, learning per-parameter preferences, and then generating designs from a visual objectives model estimated from the positive examples for use in the guided rejection sampler described in Section 4.2.

The above thought experiment exemplifies the considerations that will have to be given when combining a preliminary sampler with a refinement sampler. The two main approaches to this problem are to provide a method for generating designs that itself has tunable parameters that the refinement sampler can learn, or to use the preliminary sampler to generate a large dataset that serves as the prior for a per-parameter learning approach. It is outside the scope of this work to make a determination of what the right approach is in this instance, as additional experimentation would have to be done with both sampling methods.

The use of priors should not prevent designers from creating designs that break those principles. Experts, at least in my experience as a lighting designer, will sometimes intentionally not follow standard design principles in order to evoke a specific effect. At the moment, this functionality is retained in the systems presented here by allowing experts to use per-parameter controls, however in future interfaces it may be desireable to have a method of selectively enforcing design constraints. The exact form of such an interface would vary on a per-domain basis.

Implementing such priors in a design adjectives system may also allow for intent specification in richer terms than just a numeric score. If a designer wanted to have an image that had a minimum brightness, or wanted a form with a high amount of curvature, these intents could be encoded as priors on the design adjectives samplers. The challenge with this approach is determining how to encode the natural language of design in terms of a prior that can be interpreted by a computer system.

7.4 Parameter Selection as a First-Pass Filter

One of the simplest methods for locating desireable regions of the design space is to limit exploration to relevant parameters. Selecting a relevant subset of parameters reduces the dimensionality of the search space. The sampling methods in Chapters 4 and 5 allow users to limit which parameters get affected by the sampling process. This functions as a first-pass filter, whereby users can exclude parameters from being considered or modified if the parameters are already known to be irrelevant. In the evaluations of these systems, many users ended up not using these tools, despite asking for a way to do parameter filtering. We should be using all available tools for getting good preliminary results back quickly, so this suggests that we need a better way to indicate this functionality than we did in the prototype.

Solving this UI problem is complicated by the use of multiple selection methods during the course

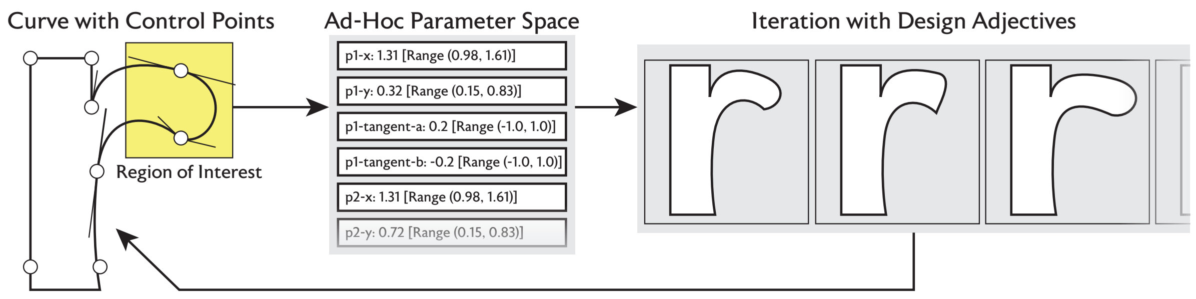

Curve with Control Points
Region of Interest
Ad-Hoc Parameter Space
p1-x: 1.31 [Range (0.98, 1.61)]
p1-y: 0.32 [Range (0.15, 0.83)]
p1-tangent-a: 0.2 [Range (-1.0, 1.0)]
p1-tangent-b: -0.2 [Range (-1.0, 1.0)]
p2-x: 1.31 [Range (0.98, 1.61)]
p2-y: 0.72 [Range (0.15, 0.83)]
Iteration with Design Adjectives

Figure 7.2: *Using Ad-Hoc Parametric Spaces with Design Adjectives.* Design adjectives can be used in non-parametric spaces by defining a subset of parameters as the design domain. In this example, a designer selects a subset of points on a curve defined by control points, and proceeds to use the design adjectives framework to iterate on the selected region of the 'r' character. Selection tools can help properly parameterize the space, in this instance, the highlighted region defines the spatial bounds for the selected points.

of an editing session. Selection is used for both editing groups of parameters, and also specifying which parameters should be affected by a sampling operation. One possible solution would be using different selection modes to indicate what operation a user is currently performing (perhaps changing the color of the highlighting). This would require the user to remember which selection mode was currently active, something that users struggled with when using the selection capabilities of the design adjectives system, which had similar functionality. Another solution would be to change the selection functionality between sampling and editing modes, creating an explicit division between generating new samples, and modifying individual parameters. Systems implementing this mode must be careful that the capabilities of each selection mode are clearly presented.

Recognizing that per-parameter selection can be useful in the exploratory design points to the connections between the design phases. Improving the low-level editing can improve the overall experience, even in parts of the process where those controls are not being used directly. While this thesis considered the three phases to be mostly distinct, improvements to one interface component may lead to unexpected improvements for other parts of the process.

7.5 Using Adjectives in Non-Parametric Spaces

The design adjectives framework is designed to work only in parameterized spaces. However, because all of the framework components work in real-time given a design domain, the framework could be used to interact with ad-hoc subsets of non-parametric design problems. This would be enabled by the ability to define a design space fulfilling the requirements of the design adjectives framework on the fly, as doing so immediately grants access to the framework's tools.

As an example, let's consider what we would need to do to enable the use of design adjectives in a glyph editor for a font (Figure 7.2). In this environment, the outlines of the glyph are represented by a curve. The curve represents its path with a list of points and maintains a list of controls needed for each point. Note that the way I have just set up this problem looks a lot like a parameterized design domain, even though curves in this environment are typically edited with non-parametric graphical tools.

If a designer in this context wanted to explore variations for a serif section of a character, such as the 'r' character in Figure 7.2, they could highlight the curve control points and instantiate a design adjectives framework using the parameters for the selected control points as the parameters for the design space. In order to bound these parameters, a range could be generated for each point based on reasonable design space priors, by assuming a range of a delta around the minimum and maximum values observed for each point, or by leveraging parameter selection tools to define reasonable bounds (for example, the highlighted region in Figure 7.2 defines the spatial parameter bounds for the selected points). From here, the design adjectives framework could be used as-written.

This hypothetical system brings up a number of interesting points to consider. First is the observation that non-parametric systems can often be represented by parameterized systems. It may not always make sense to do so (consider attempting to parameterize bitmap editing), but for the parts that can be reasonably parameterized, we can instantiate the design adjectives framework. Second, the design adjectives interface can do this ad-hoc instantiation quickly due to the stipulation that it is able to operate in real-time with no pre-existing data requirements. The models re-train in under a second, so even re-parameterizing the space can be done quickly for a single active adjective. And finally, good parameter selection and identification tools in the non-parametric application will be critical for establishing a good ad-hoc parameter space. Implementing such an ad-hoc adjectives system would be an interesting direction for future work.

7.6 Design Adjectives as a Platform

The design adjectives framework as implemented in Chapter 3 is designed for single-user desktop-based workflows. With some adjustments, the framework can be used as a platform to support workflows with other users and on other types of displays.

Multi-User Workflows Design adjectives provides a few avenues for supporting multi-user workflows. Adjectives can store presets, allowing for the development of a library that can be used by novices and experts working with a shared parameterized system. For example, the materials in the Substance Source [Sub19b] library could be augmented with adjectives that other designers created. The adjectives could then be easily customized by novice and expert designers alike.

Adjectives could also be used to obtain consensus between multiple users. In this use case, one designer would propose designs and also provide the adjective used to generate such a design. Then, a second designer could take a look and use the provided adjective to generate suggestions and update the adjective. This cycle would continue for as long as is necessary. Additional studies would have to be performed to determine if this is an effective and worthwhile use case for the design adjectives framework.

Interface Form-Factor The adjectives interface is built for desktop environments, and design tool implementers interested in bringing the interface to mobile environments, for example, would need to adjust the interface to fit the new usage environment.

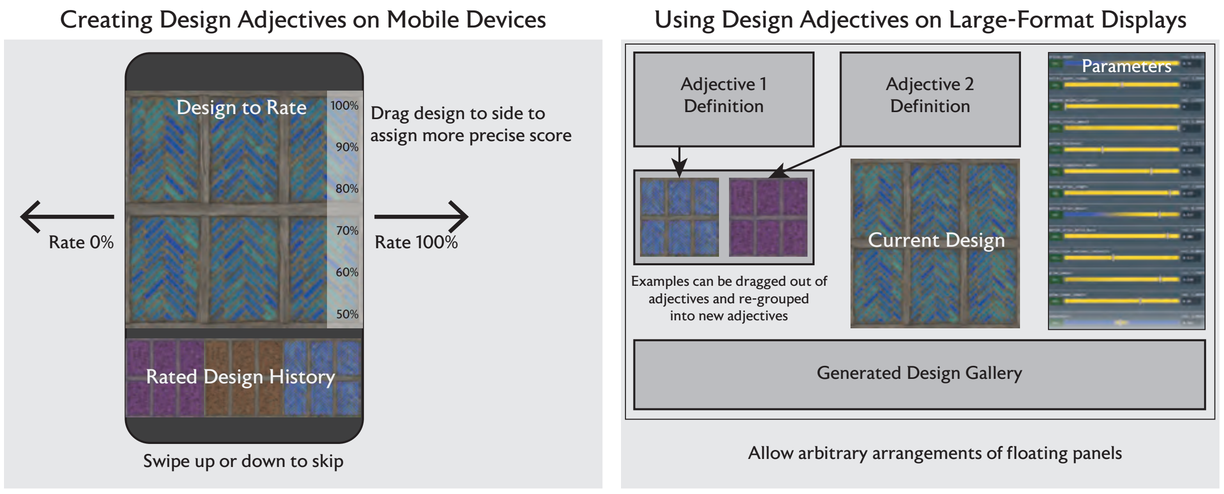

Creating Design Adjectives on Mobile Devices
Design to Rate
100%
90%
80%
70%
60%
50%
Drag design to side to assign more precise score
Rate 0%
Rate 100%
Rated Design History
Swipe up or down to skip
Using Design Adjectives on Large-Format Displays
Adjective 1 Definition
Adjective 2 Definition
Parameters
Current Design
Examples can be dragged out of adjectives and re-grouped into new adjectives
Generated Design Gallery
Allow arbitrary arrangements of floating panels

Figure 7.3: *Adapting Design Adjectives to Different Devices.* Design adjectives was developed for standard desktop environments, but could be adapted to different devices and display formats. When creating an adjective on mobile devices, a swipe-based interface could be used to allow users to rapidly rate examples to create an adjective. This type of interface would need to determine what order to present examples to the user. Large-format displays could use floating panels that can be arbitrarily arranged by a user to let the user visually arrange adjectives and designs.

For mobile devices, providing feedback for an adjective could take the form of a swipe-based interface that would allow designers to quickly bin positive and negative results. Providing more detailed scores in this interface could be achieved by holding down on the left or right of the screen and then selecting a score value (Figure 7.3-left). Returned design suggestions would still need to be arranged in a gallery, but the screen size necessarily limits the number of designs that can be shown at a time. One approach to resolve this would be to show full-size previews with a press-and-hold action on the target thumbnail.

For large-format displays, or even projected displays, it may be interesting to have the ability to visually arrange and combine adjectives (Figure 7.3-right). In this interface, adjective definitions could be spatially arranged, and examples defining each adjective freely moved and dropped to change adjective definitions, or set the active design. Combining adjectives would require some thought on the framework implementation side, as it might not be as simple as concatenating the input examples. This interface would enable designers to spatially group their ideas, allowing them to view larger relevant sections of the design space at the same time. Future studies should evaluate whether or not this interface confers benefits over the baseline desktop design adjectives interface.

7.7 Closing Comments: Computational Design Assistants

The projects in this thesis categorized tools by their role in the design process: preliminary sampling tools give an overview of the design space, refinement tools allow the designer to develop and refine a preliminary concept, and detail-oriented per-parameter controls are used to finalize the design.

These types of tools appear in multiple design domains. Building better design interfaces for specific domains involve improving one or more of these tools. Two such modifications were demonstrated: a preliminary sampling tool for theatrical lighting design, and a detail design tool for image composition editing. Creating an ideal design interface for a specific domain is a process of identifying how each of the three types of tools can best support the design process for that domain.

The functionality described here, providing tools to best assist a design task in a specific domain, is essentially what an assistant designer does. In theatrical lighting, in my experience, the assistant undertakes many of the tasks that users ask of parameterized design tools. They provide examples of preliminary lighting designs that meet the primary designer's initial, vague, design concept. They help the primary designer refine those preliminary ideas as the design goal evolves. They (sometimes) even program the lights for the primary designer, translating verbal commands to per-parameter operations. A trusted assistant is invaluable for design tasks; the designer can offload some of the creative burden of generating and testing new designs, trusting that the assistant will find something close enough to use.

The ideal interface for parameterized design should function like a design assistant. This thesis provides some methods showing how such systems would function. They provide preliminary design suggestions, they help organize and refine those preliminary designs, and they show designers what individual parameters do to the overall design. These tools should be designed to allow the designer to develop their ideas over time under an evolving set of evaluation criteria. Designers need tools to function in this way due to the human-driven constraints involved in design problems, which cannot be resolved by an optimization problem finding a single solution.

Building interfaces in this way leverages the best capabilities of the human and the computer involved in the design process. Computers are able to generate and propose designs in a matter of seconds, but are unable to evaluate the designs against a set of design aesthetics. Humans are able to instantly evaluate designs against their own design aesthetics and preferences, but have difficulty manipulating parameters to create a wide range of possible designs. Creativity support systems must be built with a consideration of the human in the design loop, using the capabilities of the computer system to enhance the creativity of the user, rather than replacing the human with a black box objective function. Computational creativity tools should assist, not replace, humans with creative tasks, and I hope that the methods presented in this thesis serve as an inspiration and a starting point for building such tools.

Appendix A

Implementations

Prototype implementations of all of the interfaces presented in this thesis can be found on the author's GitHub page: `http://github.com/ebshimizu`.

- **Design Adjectives** - `https://github.com/ebshimizu/DesignAdjectives`
- **Visual Objectives** - `https://github.com/ebshimizu/VisObjInterface`
- **Hover Visualizations** - `https://github.com/ebshimizu/sliders`

Appendix B

Preliminary Visual Objectives Evaluations

The visual objectives project underwent a series of extensive preliminary evaluations, many of which echo the findings of the main chapter. This appendix presents the preliminary studies and observations for completeness, and for those interested in how the visual objectives system came to be. These evaluations were made in context of a near-finished version of the theatrical lighting design tool described in Chapter 5.

B.1 Prior User Studies

The studies described here occurred prior to those presented in Section 5.6. The findings of these studies were used to inform the design of the evaluations used in the final system. The first two studies were run at the same time, allowing the expert evaluators for each study to score the results for both studies at the same time. Participants were allowed to take part in both studies. The third study was run after the completion of the two prior studies.

The first study is modeled after a common theatrical scenario: a designer is given a prompt and asked to quickly prepare *multiple candidate lighting designs* to present to the director of a stage production. In this scenario it is desirable to create a diverse range of results for director to review, as the hope is that at least one of these designs aligns well with the director's vision. In this study (detailed in Section B.1.1), we found that participants were always able to produce a high-scoring lighting design, according to expert evaluators, within ten minutes using the visual objectives interface. Participants did not always succeed in generating a high-scoring design using a traditional slider-per-light interface.

The second study (Section B.1.2) was patterned after the previous lighting design interface study by Kerr and Pellacini [KP09] and is designed to directly compare the performance of a lighting designer using the visual objectives interface against that of a baseline slider-per-light interface. In this study, participants used each interface to create a *single lighting design for each prompt*, and we assessed their experience using a post-study questionnaire. We had expert lighting designers evaluate the resulting images, but they often disagreed in their evaluation of the quality of a scene. Participants found the visual objectives useful but were unable to produce scenes that scored well for all evaluators using either interface, motivating the need for exploratory design to generate several candidates.

The third study (Section B.1.3) focused on the abilities of novice designers to create acceptable theatrical lighting designs using the visual objectives system. Participants were allowed to do this study if they had done the prior studies. Users were given design prompts from an introductory lighting design class, and instructed to create designs based on those prompts. They found the visual objectives system easier to use than the baseline per-light sliders interface.

B.1.1 Exploratory Design Study

In this study, we asked participants to act as the lighting designer for the director of a hypothetical theater production. To simulate a situation where a designer needs to provide quick feedback to the director, designers were only allowed to spend ten minutes lighting each scene. The goal is for the designer to produce a diverse set of designs for the director with the hope that the director highly approves of one of the designs. It is common for professional theater productions to feature over twenty scenes, so a limited amount of design time per scene is a realistic operating scenario in the early stages of development.

Experimental Setup

Tasks. Each participant was asked to complete two lighting design tasks taken from in-class exercises in an intermediate undergraduate lighting design class. For each task, the user is given a general prompt describing the scene along with environmental annotations. The two prompts used for this experiment were: *"A man sits by a fire in an open field. The fire is comforting but the overall scene feels a little creepy,"* and *"A man walks across the path at the back of the stage at dawn, feeling relaxed."* By design, these tasks are open-ended, and different color and intensity configurations can successfully satisfy the prompt. All tasks are performed on the same lighting stage featuring 190 lights and 22 light groups.

Interfaces. We compare two interfaces: the *visual objective* interface and a baseline *sliders* interface. In the sliders interface, participants select individual lights or light groups and directly specify color using a standard color-selection interface. The sliders interface is comparable to the current lighting consoles from the theater industry and was familiar to the expert participants. In order to make sure users utilized the visual objectives capability, per-light slider controls were removed from the visual objectives configuration.

Participants. The participants included three experienced theatrical and cinematic lighting designers and two novices. Each participant had normal color vision and was asked to perform two tasks. Participants started with a 20-minute tutorial session where they familiarized themselves with both interfaces. Then participants were given ten minutes with each interface to complete one of the tasks. The order of the interfaces used by each participant was randomized. Participants could produce as many designs as they desired in the allotted time.

Professional assessment. We recruited an expert lighting designer (a professor of theatrical lighting design) to serve the function of the "director" and judge the quality of the designs produced by participants. The judge was presented designs (with the corresponding task prompt) in random order and asked to assign a grade from 1 to 5 (with 5 being the best) based on how well the the design

	Visual Objectives	Sliders
Num Designs Created (avg.)	3.6	2.4
Max Task Score (avg.)	5.0	4.2
Avg Task Score (avg.)	4.3	4.0
Diversity (avg.)	5	3.4

Table B.1: *Results of exploratory design study.* All five participants were able to create a top-scoring design using the visual objectives interface. Only three of five managed to do so using the sliders interface. (Average per-participant task scores, number of designs, and diversity are averaged across all participants for each interface.)

achieved the specified goal. We refer to this grade as the *task score*. The diversity of the designs created by each participant was also rated on a 1-to-5 scale; this was used to detect cases where the artist generated a large number of similar designs.

Study Results

The results of this experiment are summarized in Table B.1. Recall that the goal of this scenario is for the director to find at least one design that they like. We found that all participants were able to generate at least one scene with the maximum rating of 5 using visual objectives interface. Only three of the five participants were able to gain a score of 5 when using the sliders interface. Table B.1 also suggest that the director determined that designs created using the visual objectives interface were more diverse than those created using the sliders interface. This greater diversity may have been useful in allowing the director to always find a design that they like.

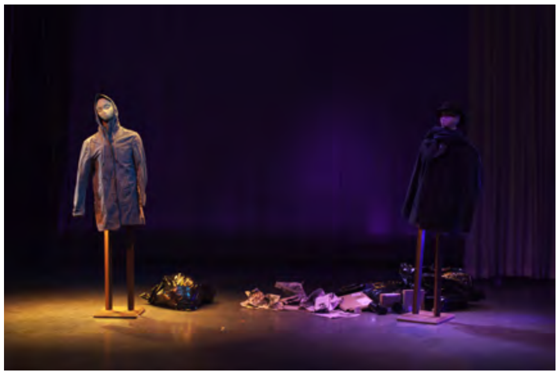
B.1.2 Interface Comparison Study

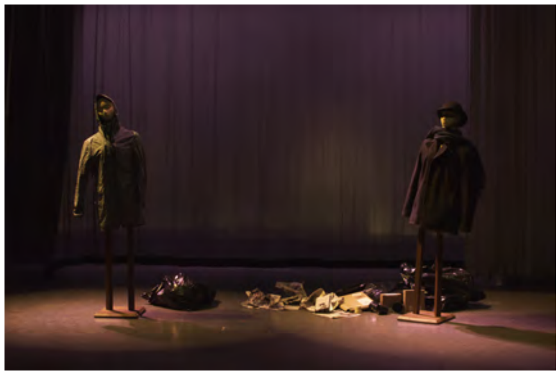
The second user study focused on assessing the performance of designers using the visual objectives interface as well as on the experience of using the system. We designed this study to closely follow prior experiments on lighting design interfaces by Kerr and Pellacini [KP09]. Participants use either the visual objectives interface and the baseline sliders interface to create a single lighting design in response to a prompt. The results are judged for task relevance and overall quality. Designers were also asked about their experience with the interfaces to assess how well our interface assists the participants' design process.

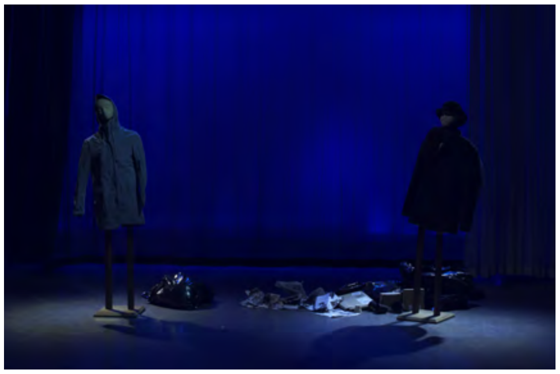
Experimental Setup

This study uses the same setup as in Section B.1.1 with the following changes. Instead of generating multiple designs, participants were asked to spend all ten minutes allotted to each task creating a single design. The study involved 12 participants, who each completed four design tasks: two with the visual objectives interface and two with the baseline sliders interface. After completing the tasks for each interface, the participants completed a survey.

The prompts for three of the tasks used in this study, along with example output from the participants is shown in Figure B.3. The fourth task was a transfer task (referred to as an *open trial* by


Judge 1's rating: 1
Judge 2's rating: 5


Judge 1's rating: 4
Judge 2's rating: 1


Judge 1's rating: 4
Judge 2's rating: 1

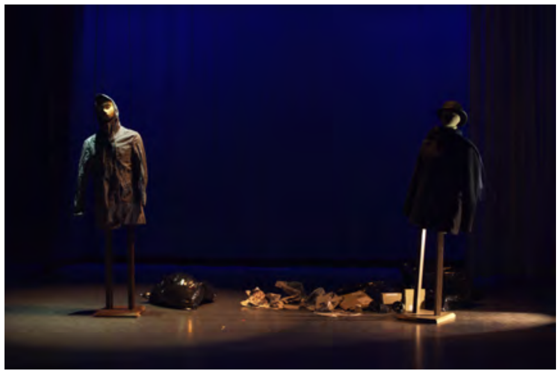

Judge 1's rating: 5
Judge 2's rating: 2

Figure B.1: Four results from our interface comparison study for task (A) in Figure B.3. Due to individual preferences for particular lighting designs, expert judges showed strong disagreement in assessing the quality of the scenes.

Kerr and Pellacini [KP09]). In this task, the participant is presented with a photograph from a theatrical scene and asked to transfer the lighting configuration in the photograph to the target stage. Participants were not allowed to use the photograph in the visual objectives system. This task is more constrained since the artist's choice of colors and relative intensities is intended to follow the input photograph as closely as possible. All tasks are performed on the same lighting stage, but different stage props are used in the three scenarios to contextualize the task (Figure B.3). The stage features 44 lights and 13 light groups.

To judge the quality of resulting designs, we recruited an additional professional lighting designer (to augment the judge used in the first study) and asked both judges to evaluate each design's *task score* (relevance to task) and *quality score* (the overall quality of the lighting without regard to the task).

Study Results

Figure B.3 provides a sampling of designs created by participants during the user study, and shows the visual objectives used to achieve these designs.

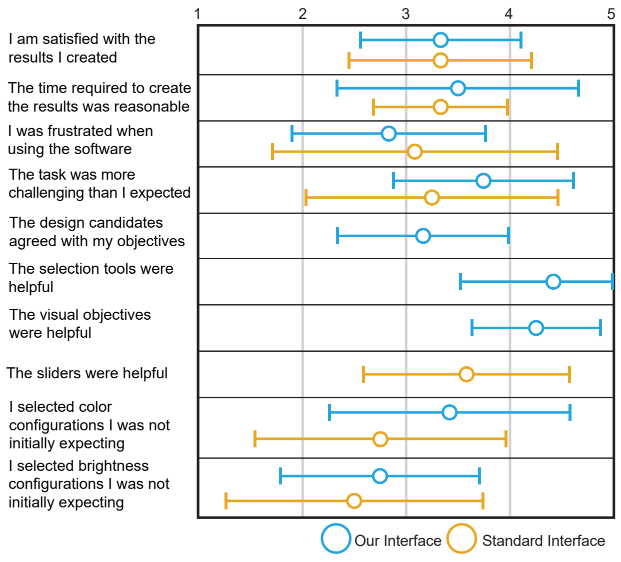


Figure B.2: Average participant response to each question on the interface comparison study questionnaire in Section B.1.2. Responses are on a 1-to-5 scale, with 1 indicating disagreement, and 5 agreement. Error bars represent one standard deviation.

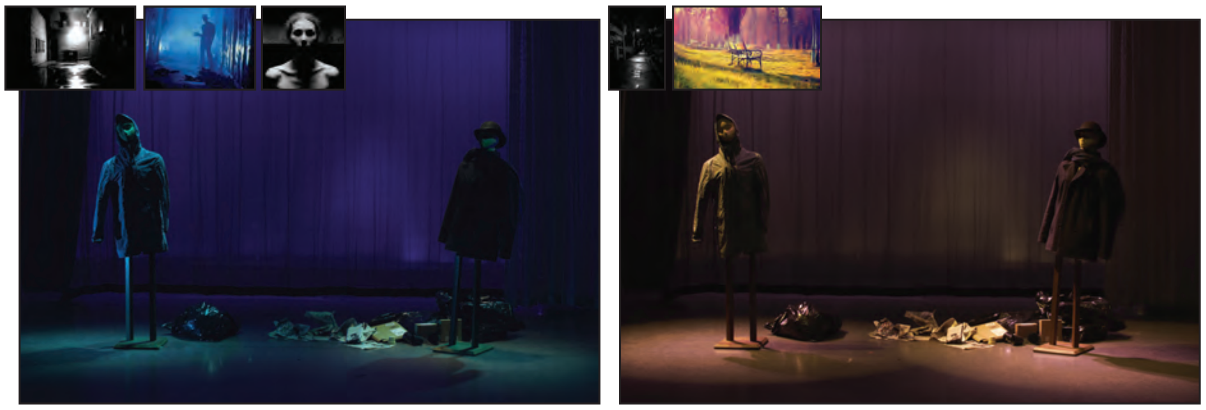
	Visual Objectives	Sliders
Avg Task Score	3.1	3.3
Avg Quality Score	3.0	3.2

Table B.2: *Results of interface comparison study.* Average scores for the two interfaces in our interface comparison (Section B.1.2) study. All results were evaluated on a 1-to-5 Likert scale (higher is better).

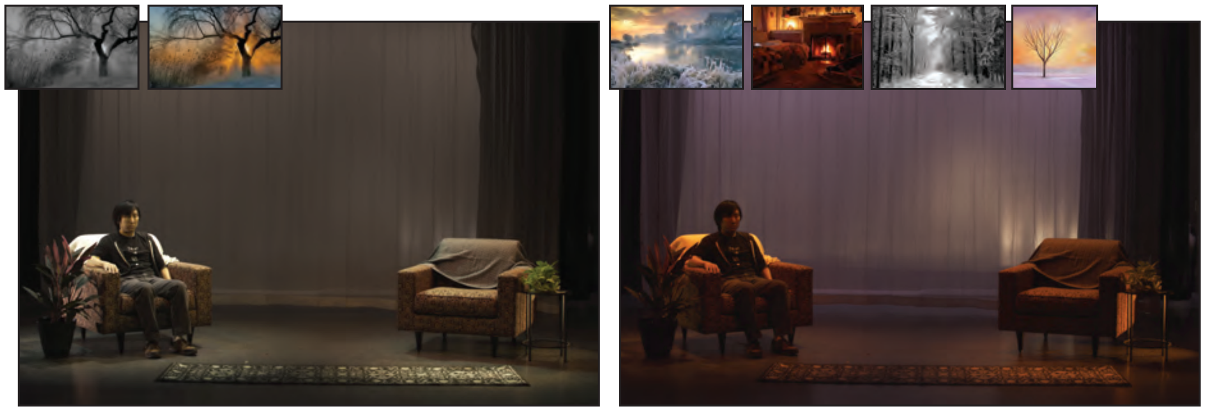
Observations and Participant Experiences. The results of the user study questionnaire are displayed in Figure B.2. Participants found visual objectives useful when performing the design tasks (avg. 4.25) and all participants had positive experiences with the UI for targeting visual objectives on stage (avg. 4.42). Responses suggest that participants found the design candidates generated by the system to be a useful part of their design process, even though sometimes these candidates did not line up with participant expectations (avg. 3.16). This could be viewed as a failure of sampling to capture visual objectives in some instances, but also could be viewed as a useful property that enables participants to encounter design candidates that would not have otherwise been considered.

Participants indicated that they were less frustrated using the visual objectives interface when performing tasks (avg. 2.83) compared to the baseline (avg. 3.08), but not to a significant degree. They also reported that accomplishing tasks was slightly more difficult than expected using visual objectives (avg. 3.75 vs avg. 3.25 for the baseline), also not to a significant degree. This may be due to unfamiliarity with the visual objectives interface, or that participants simply expected the interface to be easier to use after the tutorial session.

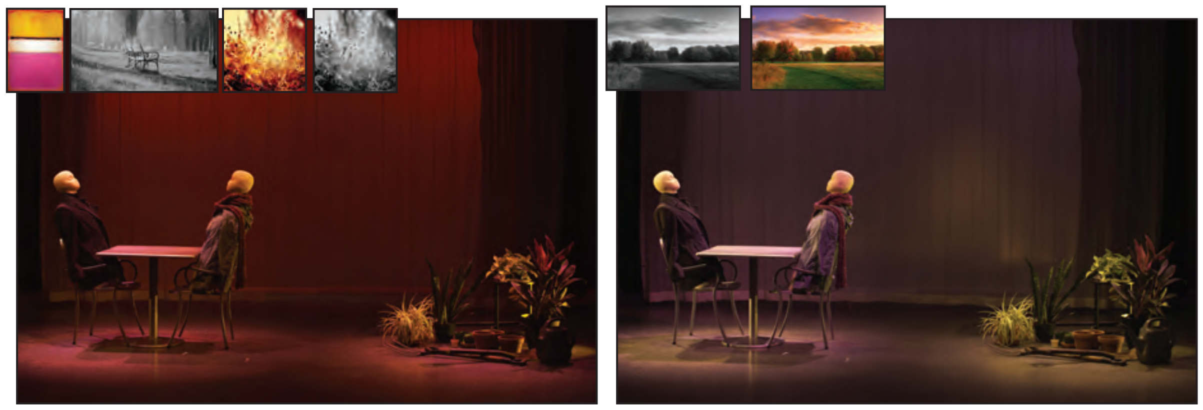
When working with the interface, we observed that participants typically began applying a color and intensity objective to the entire stage in order to quickly get to a point close to their intended design. If the objectives did not result in the desired effect, participants often chose a different set of visual objectives and repeated the process. Participants were able to reach a design that they liked



(a) It is a summer night in the city. A suspicious man dressed in dark clothes steps out of an alley ready to follow the woman who is passing by.



(b) It is a cold winter morning. People have not yet begun the day. A man sitting in an armchair before a fire, looks out the window and enjoys the silence.



(c) It is a late spring evening. A couple sits at a table in their garden watching the sunset.

Figure B.3: Designs created by participants of the interface comparison (Section B.1.2) study using the visual objectives interface. The reference images used by the participants to create their designs are shown in the top left of each design.

quickly but expressed a desire to use traditional sliders to perform small modifications on the scene to perfect it; as expected this desire was especially prevalent among experienced participants familiar with direct light manipulation interfaces. More detailed observations of the participants can be found in the supplemental materials.

Professional Assessment. Two expert lighting designers judged the quality of the participant's designs (results summarized in Table B.2). In contrast to the first user study, overall task score for both interfaces, *when results are average across both judges*, was low—at most 3.3 (similar results hold for the quality score as well). Further examination revealed this lower average is due to high disagreement between the two expert judges. (The variance in task score was 0.9 on our 1-to-5 scale.) Although both expert judges assigned some scenes high ratings, they were often inconsistent in the designs they preferred. Given a condition like *"summer night in a city"*, the judges did not agree over an acceptable range of colors and intensities, and were critical of designs that were inconsistent with their expectations. Figure B.1 provides examples where the judges disagreed by a spread of 4 or more.

While these results make it difficult to draw quantitatively conclusions about the performance of participants in the study, we believe they (along with the results of the first user study) suggest that there is notable value in the proposed exploratory design interface. The ability to rapidly explore different designs is valuable aid for both individual creative thought and also communication between creative professionals.

B.1.3 Novice Designer Study

Our third user study examined the extent to which the proposed interface could assist novice lighting designers in their design process. We recruited users with little to no lighting design experience and asked them to create designs for two different scenarios. Users were given as much time as they wanted to create the design.

Experimental Setup

Each participant was asked to complete two lighting design tasks taken from in-class exercises in an intermediate undergraduate lighting design class. For each task, the user is given a general prompt describing the scene along with environmental annotations. The two prompts used for this experiment were: *"It is a late spring evening. A couple sits at a table in their garden watching the sunset.,"* and *"It is a cold winter morning. People have not yet begun the day. A man sitting in a armchair before a fire, looks out the window and enjoys the silence."* By design, these tasks are open-ended, and different color and intensity configurations can successfully satisfy the prompt. All tasks are performed on the same lighting stage featuring 44 lights and 13 light groups (shown in Figure B.5).

Participants were trained to use two different lighting design interfaces: a traditional slider-based interface where each light parameters could be adjusted using slider controls, and our visual objectives interface. In the sliders interface, participants select individual lights or light groups and directly specify color using a standard color-selection interface. The sliders interface is representative of current lighting consoles from the theater industry. The chosen interface for each task was randomized for each participant.

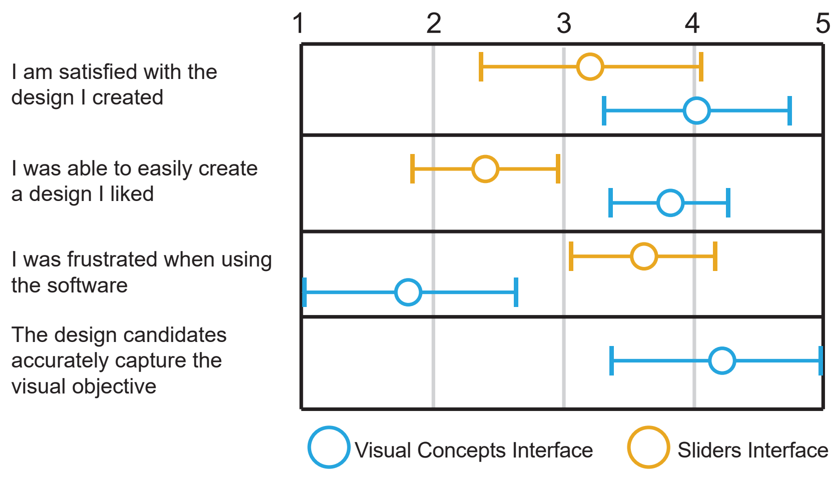


Figure B.4: Average participant response to each question on the novice study questionnaire in Section B.1.3. Responses are on a 1-to-5 scale, with 1 indicating disagreement, and 5 agreement. Error bars represent one standard deviation.

We recruited five participants with little to no lighting design experience for this study. Participants were given as much time as needed to complete the lighting design tasks. Participants were also given a pre-selected library of reference images, but were allowed to find images on the internet if they desired (only one participant searched for their own images). After completing both tasks, participants completed a short survey where they rated the self-assessed quality of their design, along with their experience using each of the interfaces.

Results

While participants ended up with scenes that they were satisfied with in both interfaces, they reported a much higher level of frustration with the baseline sliders interface (Figure B.4). For novice users, much of the frustration with the sliders interface comes from being unable to predict how changing one set of lights will affect the entire scene, leading to a tedious process of trial-and-error as they attempt to adjust light parameters to match their intended design. With the visual objectives interface, this process was made much easier and faster, since changing a design idea was as simple as selecting a different image from the image library.

Novice participants agreed that the system produced good design candidates from their selected images. Most users left model parameters extracted from reference images alone, however one user adjusted the color objective to contain colors that were not present in the original image, suggesting that none of the available research images exactly matched their ideal color palette. Rather than finding another reference image, this participant preferred to manually set their own palette. Some of the scenes created with our interface are shown in Figure B.5.

(a) It is a late spring afternoon/evening. A person walks through a garden as the sun sets.

(b) It is a cold winter morning. One person goes about their morning tasks in their home.

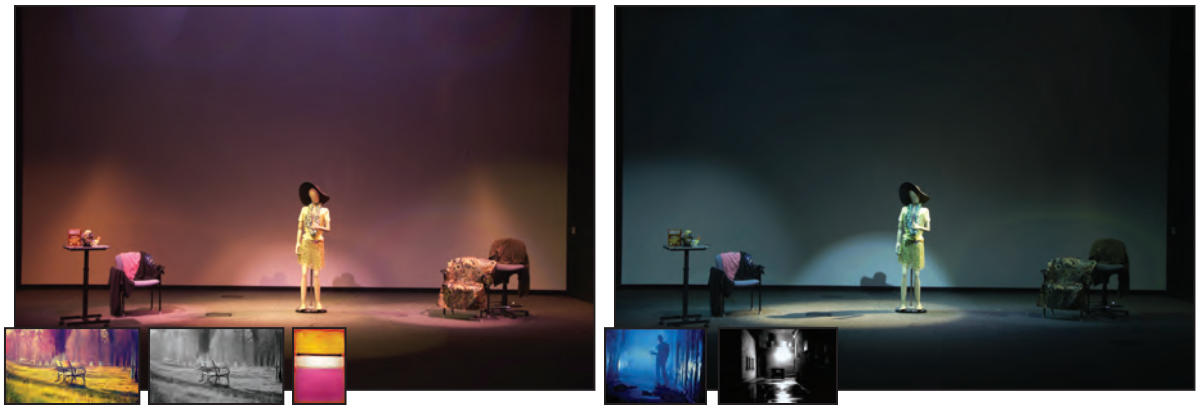


Figure B.5: Designs created by participants of the novice study (Section B.1.3) using the visual objectives interface. The reference images used by the participants to create their designs are shown in the top left of each design.

B.2 Workflow Observations

We had a lot of time to observe users during the studies listed above. Users demonstrated a variety of different methods of working with the visual objectives interface, varying how they used the image-defined objectives, selection methods, and when they decided to use per-parameter controls. Some of the common workflows observed are described here.

Primary Concepts. One of the first operations performed by almost all users of our interface is to pick two objectives, a color objective and intensity objective (often from the same image, but not always), and apply those two objectives to the entire stage. These objectives serve as the primary objectives used in the scene. If the results do not quite line up with what the user is looking for, they would either find a new image or directly adjust the model parameters, and re-run the sampling method for the objective. Once the stage looks roughly in line with their expectations, they remove the primary objectives and start refining their design. Users typically then pick a different objective to apply to a smaller region of the stage to further refine their design.

Targeting and Selection. Many users chose to use inclusive selection to apply objectives, instead of excluding lights from being affected by the objective. This is likely due to the small size of the light lab configuration used in this preliminary evaluation, where every area can be cleanly isolated. Some users also had some difficulty intuiting what lights would be selected when a visual objective was used. The difficulty comes from the fact that physical lights tend to spill outside of a particular selected region onto the floor or other background elements, and the selected lights are those that *influence* the selected region, not *contained within* the selected region. We provided users with a view that displayed only the selected lights, similar to a hover visualization (Chapter 6), to help visualize how selection works.

Fine Tuning. One configuration tested with preliminary users removed the per-parameter slider controls from the interface. This proved to be rather unpopular. Users were often able to quickly get to the point with our interface that they wanted to start doing detailed adjustments to individual lights. The speed at which users reached this point is indicative of the effectiveness of our system, as it allows expert designers to quickly start doing tasks that require their expertise instead of spending those first few minutes just setting up the scene, and it allows intermediate users to create better designs more quickly.

As a result of the speed of the interface, users actually spent *less* time using the objectives by themselves than they did tweaking the results generated from the objectives. This is a positive indication that the interface we built accomplishes the goals of effectively exploring a reasonable space of designs as defined by the user's objectives by getting them to a point where they can perform fine-tuning operations quickly.

Layering Objectives. Many users chose to work with one objective at a time rather than using multiple objectives at the same time, and some were confused by the need to manipulate intensity and color concepts separately. The confusion may stem from some UI problems found in the version of the tool used in the study, but also suggests that people can only hold a few concepts in their mind when performing a complex design task. Keeping the list of active concepts displayed in the interface helps with keeping track of references while designing, and was added to the final version of the interface presented in Chapter 5.

Appendix C

Image Composition Rendering Model

The renderer used in the hover visualizations project treated an image composition as a parameterized design space. This was accomplished by using a fixed set of adjustment layers and layer transparency to define the input parameter vector. This appendix describes the rendering model used for the project, and describes an experimental, and ultimately unused, adjustment made to layer grouping called "lists."

C.1 Group Rendering Model

Current image manipulation software places layers into a rendering graph, which is often displayed as a list which organizes the layers by depth. Conceptually, the layers sit on transparent planes that get composed with the layers beneath them following the rules defined by the blend mode function of the layer. Some layers, called adjustment layers, have no pixels data associated with them, but instead apply a filter (brightness, contrast, hue, etc.) on the current state of the composition. Rendering proceeds from bottom-up, composing the current state of the composition with the next layer in the graph.

In this model, layers (including adjustment layers), can be organized into groups, typically represented as folders in the layer list display. Groups can either be treated as *pass-through*, meaning that they are only used for semantic grouping and do not affect the render graph order, or they can be given a blend mode and will be treated as a *precomposition* operation, as shown in Figure C.1. When a precomposition group is encountered, the renderer will first compose the layers in the group as if they layers were a new, separate, composition. The result of that operation is then composed with the current state of the composition using the blend mode and opacity setting of the group. When a pass-through group is encountered, the layers are blended with the current composition starting with the lowest depth ordered layer in the group, modulated by the group's opacity value. Groups may also have adjustment layers applied to them. Semantically similar layers are often kept on different depths or within different groups, as highlighted in red on Figure C.1. This makes it difficult to perform batch edits and maintain consistent adjustment settings, as only one layer's settings can be adjusted at a time.

To render the composition, the layers are processed from bottom to top. Each rendering step

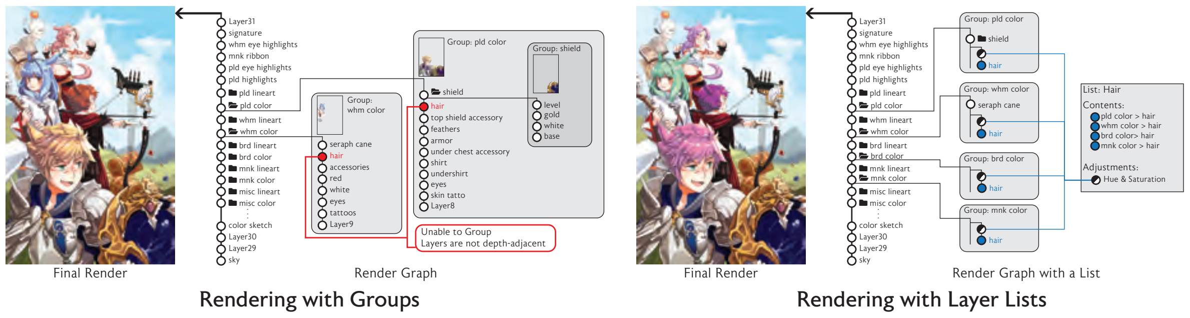


Figure C.1: *Rendering a Composition with Groups and Lists.* The rendering graph for each composition is shown to the right. Nodes with a folder icon are groups using the default normal blending mode. Rendering proceeds from bottom to top. Note that groups can only consist of depth-adjacent layers, and it is impossible to group the highlighted "hair" layers (left), as they are not depth-adjacent. With an explicit layer list, (right), the hair layers located in different groups for rendering can all be adjusted with a single HSV adjustment layer. The effect of the HSV adjustment is shown in the final composition.

takes the current state of the composition and the next highest layer in depth order, and composites the two together according to the layer's blend mode. The next state of the composition c_{n+1} can thus be described as:

$$c_{n+1} = l_{n+1} \circ_{n+1} c_n \tag{C.1}$$

where l_{n+1} is the next layer above the composition c_n and $\circ_{n+1}$ is the layer's blend mode operation. All layers are assumed to have an opacity value and a blend mode.

These systems also support adjustment layers, which are functions, a_{n+1} that perform per-pixel operations on layer pixels according to their adjustment settings. These adjustment layers are used in two ways: as global adjustment layers that change all currently rendered pixels below where the adjustment layer is placed in the depth ordering (operating on c_n), or as local adjustment layers which operate only on an individual layer or non-pass-through group (Figure C.2). If the adjustment is run on the state of the composition, the result of that adjustment operation can have its own blend mode and transparency settings. With adjustment layers, the next state of the composition can now be described with the following rules:

$$c_{n+1} = a_{n+1}(c_n) \circ_{n+1} c_n \qquad \text{for global adjustment layers} \tag{C.2}$$

$$c_{n+1} = a_{n+1}(l_{n+1}) \circ_{n+1} c_n \qquad \text{for local adjustment} \tag{C.3}$$

In the case where there are multiple local adjustments on a single layer, the adjustments can be composed into a function $d'_n = d^m_n(\ldots d^2_n(a^1_n(l_n))\ldots)$ which is then applied to the layer before composition.

C.2 List Rendering Model

An experimental grouping model for layers was implemented in the hover visualization interface, but ultimately went un-used. The model was developed to experiment with providing different methods of layer grouping, unconstrained by the tree structure of the group rendering model. Lists are sets of

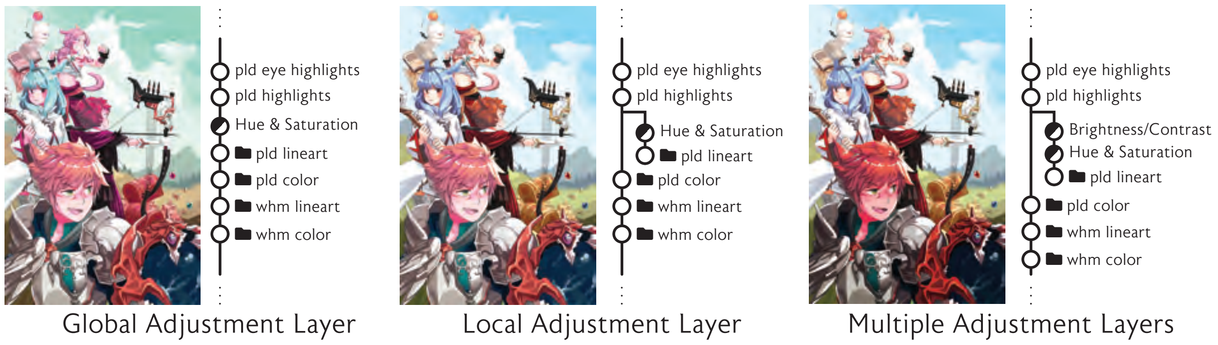


Figure C.2: *Adjustment Layers.* Adjustment layers modify the layer or composition pixels. A global adjustment layer affects all layers below it (left). A local adjustment layer only affects the layer or non-pass-through group immediately below it (center). Multiple adjustments can be applied in series following the normal bottom-to-top composition order (right).

layers that exist outside of the rendering depth order. Lists provide the ability to group and manipulate layers without requiring them to be in render groups that consist of depth-adjacent layers. Lists retain the same functionality that groups have, with the exception of the blend mode functionality, as lists do not affect the structure of the render graph.

In order to attain parity with layer group adjustments, lists simply distribute adjustments across the layers that they contain, and apply opacity changes multiplicatively, the same way that pass-through groups do. When a layer is affected by a list that has an adjustment applied to it, the layer's adjustment function is composed with the list's adjustment function as described in Section C.1. Note that the layer's order of adjustments can be edited such that the list adjustments apply before, or are interleaved with, the layer's individual adjustments. We choose to hide this functionality in our implementation of the interface to reduce the interface complexity. The rendering system makes no attempt to combine adjustments of the same type, as many of them are non-commutative. Lists were able to replicate functionality similar to Local Layering [MP09], providing a possible method of implementing that method within existing layering systems with a small number of modifications.

Bibliography

[Aca20] Academy Software Foundation. openexr, 2020. URL: `https://github.com/AcademySoftwareFoundation/openexr`. 5.5.2

[ADA$^+$04] Aseem Agarwala, Mira Dontcheva, Maneesh Agrawala, Steven Drucker, Alex Colburn, Brian Curless, David Salesin, and Michael Cohen. Interactive digital photomontage. *ACM Transactions on Graphics (TOG)*, 23(3):294–302, August 2004. doi:10.1145/1015706.1015718. 5.2

[Ado20] Adobe Systems. Adobe Photoshop, 2020. URL: `https://www.adobe.com/products/photoshop.html`. (document), 6.3, 6.2

[ADW04] Frederik Anrys, Philip Dutré, and Yves D. Willems. Image-based lighting design. In *Proceedings of the 4th IASTED International Conference on Visualization, Imaging, and Image Processing*, pages 15–15, 2004. URL: `http://graphics.cs.kuleuven.be/publications/IBLD/IBLD_paper.pdf`. 5.2

[ALK$^+$03] D. Akers, F. Losasso, J. Klingner, M. Agrawala, J. Rick, and P. Hanrahan. Conveying shape and features with image-based relighting. In *IEEE Visualization, 2003. VIS 2003.*, pages 349–354, October 2003. doi:10.1109/VISUAL.2003.1250392. 5.2

[Are02] ArenaNet. Guild Wars 2, August 2002. 1

[art19] artinsomiacs. Art Insomniacs on Twitter: "I am a: Man Woman Artist Looking for: Man Woman The layer I'm supposed to be drawing on" / Twitter, August 2019. URL: `https://twitter.com/artinsomniacs/status/1162393581296001024`. 6

[Aut] Autodesk. nCloth | Maya 2019. URL: `https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2019/ENU/Maya-CharEffEnvBuild/files/GUID-ED791F1C-8412-4785-829F-9925F2604E8A-htm.html`. 1

[BAN18] BANDAI NACO Entertainment. SOULCALIBUR VI, October 2018. 1

[Ban19] Bandai Namco Studios. Code Vein, September 2019. 1

[Bay04] Nigan Bayazit. Investigating Design: A Review of Forty Years of Design Research. *Design Issues*, 20(1):16–29, January 2004. URL: `http://www.mitpressjournals.org/doi/10.1162/074793604772933739`, doi:10.1162/074793604772933739. 2.1

[BBdF10] Eric Brochu, Tyson Brochu, and Nando de Freitas. A Bayesian Interactive Optimization Approach to Procedural Animation Design. page 10, 2010. 2.2.2

[Bel57] Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957. Google-

Books-ID: wdtoPwAACAAJ. 2.2

[Bet15] Bethesda Game Studios. Fallout 4, November 2015. 1

[BG04] Patrick Baudisch and Carl Gutwin. Multiblending: displaying overlapping windows simultaneously without the drawbacks of alpha blending. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '04, pages 367–374, Vienna, Austria, April 2004. Association for Computing Machinery. doi:10.1145/985692.985739. 2.2.6, 6.3

[BHU$^+$19] Glen Berseth, Brandon Haworth, Muhammad Usman, Davide Schaumann, Mahyar Khayatkhoei, Mubbasir Turab Kapadia, and Petros Faloutsos. Interactive Architectural Design with Diverse Solution Exploration. *IEEE Transactions on Visualization and Computer Graphics*, pages 1–1, 2019. doi:10.1109/TVCG.2019.2938961. 2.2.5

[Bio07] BioWare. Mass Effect, November 2007. 1

[Bio09] BioWare. Dragon Age: Origins, November 2009. 1

[Bli04] Blizzard Entertainment. World of Warcraft, November 2004. 1

[BPB13] Ivaylo Boyadzhiev, Sylvain Paris, and Kavita Bala. User-assisted image compositing for photographic lighting. In *ACM Transactions on Graphics*, volume 32, page 1, July 2013. URL: http://dl.acm.org/citation.cfm?doid=2461912.2461973, doi:10.1145/2461912.2461973. 2.2.5, 5.2

[BSP$^+$93] Eric A. Bier, Maureen C. Stone, Ken Pier, William Buxton, and Tony D. DeRose. Toolglass and magic lenses: the see-through interface. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques - SIGGRAPH '93*, pages 73–80. ACM Press, 1993. URL: http://portal.acm.org/citation.cfm?doid=166117.166126, doi:10.1145/166117.166126. 2.2.6, 6.3

[Bux10] Bill Buxton. *Sketching User Experiences: Getting the Design Right and the Right Design.* Morgan Kaufmann, July 2010. Google-Books-ID: 2vfPxocmLh0C. 2.1.2

[Cap12] Capcom. Dragon's Dogma, May 2012. 1

[Cap18] Capcom. Monster Hunter: World, January 2018. 1

[CEL20] CELSYS. Clip Studio Paint, 2020. URL: https://www.clipstudio.net/en/. (document), 6.3, 6.2

[CG92] George Casella and Edward I. George. Explaining the Gibbs Sampler. *The American Statistician*, 46(3):167–174, 1992. URL: https://www.jstor.org/stable/2685208, doi:10.2307/2685208. 5.1, 5.4.2

[CKGF13] Siddhartha Chaudhuri, Evangelos Kalogerakis, Stephen Giguere, and Thomas Funkhouser. Attribit: content creation with semantic attributes. In *Proceedings of the 26th annual ACM symposium on User interface software and technology - UIST '13*, pages 193–202, St. Andrews, Scotland, United Kingdom, 2013. ACM Press. URL: http://dl.acm.org/citation.cfm?doid=2501988.2502008, doi:10.1145/2501988.2502008. 2.2.1, 2.2.2

[CL14] Erin Cherry and Celine Latulipe. Quantifying the Creativity Support of Digital Tools through the Creativity Support Index. In *ACM Transactions on Computer-Human Interaction*, volume 21, pages 1–25, June 2014. URL: `http://dl.acm.org/citation.cfm?doid=2633907.2617588`, `doi:10.1145/2617588`. 5.6, 5.6.3

[CP04] Cryptic Studios and Paragon Studios. City of Heroes, April 2004. 1

[Cro11] Nigel Cross. *Design thinking: understanding how designers think and work*. Berg, Oxford, engl. ed edition, 2011. OCLC: 846312366. 2.1.2

[Cry10] Cryptic Studios. Star Trek Online, February 2010. 1

[CS08] Juliet Corbin and Anselm Strauss. *Basics of Qualitative Research (3rd ed.): Techniques and Procedures for Developing Grounded Theory*. SAGE Publications, Inc., 2455 Teller Road, Thousand Oaks California 91320 United States, 2008. URL: `http://methods.sagepub.com/book/basics-of-qualitative-research`, `doi:10.4135/9781452230153`. 5.6.1

[Csi90] Mihaly Csikszentmihalyi. *Flow: the psychology of optimal experience*. Harper & Row, New York, 1st ed edition, 1990. 1.1

[CWA14] Patricia Cohen, Stephen G West, and Leona S Aiken. *Applied multiple regression/correlation analysis for the behavioral sciences*. Psychology Press, 2014. 2.2.2

[DAG95] J. Dorsey, J. Arvo, and D. Greenberg. Interactive design of complex time dependent lighting. *IEEE Computer Graphics and Applications*, 15(2):26–36, March 1995. `doi:10.1109/38.365003`. 5.2

[DAM+19] Ruta Desai, Fraser Anderson, Justin Matejka, Stelian Coros, James McCann, George Fitzmaurice, and Tovi Grossman. Geppetto: Enabling Semantic Design of Expressive Robot Behaviors. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, pages 369:1–369:14, New York, NY, USA, 2019. ACM. event-place: Glasgow, Scotland Uk. URL: `http://doi.acm.org/10.1145/3290605.3300599`, `doi:10.1145/3290605.3300599`. 2.2.2

[Dar79] Jane Darke. The primary generator and the design process. *Design Studies*, 1(1):36–44, July 1979. URL: `http://linkinghub.elsevier.com/retrieve/pii/0142694X79900279`, `doi:10.1016/0142-694X(79)90027-9`. 2.1.2

[Dor06] Kees Dorst. Design Problems and Design Paradoxes. *Design Issues*, 22(3):4–17, July 2006. URL: `http://www.mitpressjournals.org/doi/10.1162/desi.2006.22.3.4`, `doi:10.1162/desi.2006.22.3.4`. 2.1.2

[DSG91] Julie O'B. Dorsey, Françis X. Sillion, and Donald P. Greenberg. Design and Simulation of Opera Lighting and Projection Effects. In *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '91, pages 41–50, New York, NY, USA, 1991. ACM. URL: `http://doi.acm.org/10.1145/122718.122723`, `doi:10.1145/122718.122723`. 5.2

[EE19] EA Vancouver and EA Romania. FIFA 20, September 2019. 1

[Ele20] Electron. Electron, 2020. URL: `https://www.electronjs.org/`. 4.5

[Epi] Epic Games. Unreal Engine. URL: `https://www.unrealengine.com/en-US/`. 1

[ETC20] ETC. Eos Family, 2020. URL: https://www.etcconnect.com/Products/Consoles/Eos-Family/. 1, 2.2.6, 5.5.3

[Fac20] Facebook. Instagram, 2020. URL: https://instagram.com/. 1

[FEHF09] Ali Farhadi, Ian Endres, Derek Hoiem, and David Forsyth. Describing objects by their attributes. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1778–1785, June 2009. ISSN: 1063-6919. doi:10.1109/CVPR.2009.5206772. 2.2.2

[FF93] Martin R. Frank and James D. Foley. Model-based user interface design by example and by interview. In *Proceedings of the 6th annual ACM symposium on User interface software and technology*, UIST '93, pages 129–137, Atlanta, Georgia, USA, December 1993. Association for Computing Machinery. doi:10.1145/168642.168655. 4.6.1

[Fou20] Foundry. Katana, 2020. URL: https://www.foundry.com/products/katana. 1

[F.R01] Karl Pearson F.R.S. LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, November 1901. doi:10.1080/14786440109462720. 2.2.2

[Fro15] FromSoftware. Bloodborne, March 2015. 1

[FRS+12] Matthew Fisher, Daniel Ritchie, Manolis Savva, Thomas Funkhouser, and Pat Hanrahan. Example-based Synthesis of 3D Object Arrangements. In *ACM Trans. Graph.*, volume 31, pages 135:1–135:11, November 2012. URL: http://doi.acm.org/10.1145/2366145.2366154, doi:10.1145/2366145.2366154. 2.2.1

[FSdST+11] C.D. Ferreira, J.A. Santos, R. da S. Torres, M.A. Gonçalves, R.C. Rezende, and Weiguo Fan. Relevance feedback based on genetic programming for image retrieval. In *Pattern Recognition Letters*, volume 32, pages 27–37, January 2011. URL: http://linkinghub.elsevier.com/retrieve/pii/S0167865510001558, doi:10.1016/j.patrec.2010.05.015. 2.2.4

[FZ08] Vittorio Ferrari and Andrew Zisserman. Learning Visual Attributes. In J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 433–440. Curran Associates, Inc., 2008. URL: http://papers.nips.cc/paper/3217-learning-visual-attributes.pdf. 2.2.2

[Gar19] Vincent Garreau. particles.js, October 2019. original-date: 2019-10-01T18:30:48Z. URL: https://github.com/ebshimizu/particles.js. 1, 4.5.3, 4.6.1

[GBH09] Tovi Grossman, Patrick Baudisch, and Ken Hinckley. Handle Flags: efficient and flexible selections for inking applications. In *Proceedings of Graphics Interface 2009*, GI '09, pages 167–174, Kelowna, British Columbia, Canada, May 2009. Canadian Information Processing Society. 6.3

[Gol] Golaem. Golaem. URL: http://golaem.com/content/evaluate-sdk. 1

[GP92] Vinod Goel and Peter Pirolli. The structure of Design Problem Spaces. *Cognitive Science*, 16(3):395–429, 1992. URL: https://onlinelibrary.wiley.com/doi/abs/10.1207/s15516709cog1603_3, doi:10.1207/s15516709cog1603_3. 1.1, 2.1.2

[GPB+18] Jacob R. Gardner, Geoff Pleiss, David Bindel, Kilian Q. Weinberger, and Andrew Gordon Wilson. GPyTorch: Blackbox Matrix-Matrix Gaussian Process Inference with

GPU Acceleration. *arXiv:1809.11165 [cs, stat]*, September 2018. arXiv: 1809.11165. URL: http://arxiv.org/abs/1809.11165. 4.1, 4.5

[Hep03] Daryl H Hepting. Interactive Evolution for Systematic Exploration of a Parameter Space. In *Proceedings of ANNIE 2003*, pages 125–131, 2003. URL: http://www2.cs.uregina.ca/~hepting/research/dwnld/annie2003interactevo.pdf. 2.2.4

[HG05] Daryl H. Hepting and David Gerhard. Collaborative Computer-Aided Parameter Exploration for Music and Animation. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Dough Tygar, Moshe Y. Vardi, Gerhard Weikum, and Uffe Kock Wiil, editors, *Computer Music Modeling and Retrieval*, volume 3310, pages 158–172. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. URL: http://link.springer.com/10.1007/978-3-540-31807-1_13, doi:10.1007/978-3-540-31807-1_13. 2.2.4

[HGMS04] Daryl H. Hepting, David Gerhard, Matthew McKague, and Paul Schmiedge. Managing parameter spaces for multimedia composition. In *ACM SIGGRAPH 2004 Posters on - SIGGRAPH '04*, page 78, Los Angeles, California, 2004. ACM Press. URL: http://portal.acm.org/citation.cfm?doid=1186415.1186505, doi:10.1145/1186415.1186505. 2.2.4

[HJO$^+$01] Aaron Hertzmann, Charles E. Jacobs, Nuria Oliver, Brian Curless, and David H. Salesin. Image analogies. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques - SIGGRAPH '01*, pages 327–340. ACM Press, 2001. URL: http://portal.acm.org/citation.cfm?doid=383259.383295, doi:10.1145/383259.383295. 2.2.3

[HLCO10] Hai Nam Ha, Christophe Lino, Marc Christie, and Patrick Olivier. An Interactive Interface for Lighting-by-Example. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Robyn Taylor, Pierre Boulanger, Antonio Krüger, and Patrick Olivier, editors, *Smart Graphics*, volume 6133, pages 244–252. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. URL: http://link.springer.com/10.1007/978-3-642-13544-6_23, doi:10.1007/978-3-642-13544-6_23. 2.2.3

[HYP$^+$10] Ken Hinckley, Koji Yatani, Michel Pahud, Nicole Coddington, Jenny Rodenhouse, Andy Wilson, Hrvoje Benko, and Bill Buxton. Pen + touch = new tools. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology*, UIST '10, pages 27–36, New York, New York, USA, October 2010. Association for Computing Machinery. doi:10.1145/1866029.1866036. 6.3

[ice19] icedcitruss. icedcitruss on Twitter: "do people name their layers when they draw? i'm just a feral bastard who draws with like 20 layers and clicks thru each one trying to find the right one." / Twitter, October 2019. URL: https://twitter.com/icedcitruss/status/1186023950955859968. 6, 6.1

[IF04] Edward W. Ishak and Steven K. Feiner. Interacting with hidden content using content-aware free-space transparency. In *Proceedings of the 17th annual ACM symposium on User interface software and technology - UIST '04*, page 189, Santa Fe, NM, USA, 2004. ACM Press. URL: `http://portal.acm.org/citation.cfm?doid=1029632.1029666`, `doi:10.1145/1029632.1029666`. 2.2.6, 6.3

[Kai92] Kai Krause. Kai's Power Tools, 1992. 1

[KCLK14] Ashish Kapoor, Juan C. Caicedo, Dani Lischinski, and Sing Bing Kang. Collaborative Personalization of Image Enhancement. *International Journal of Computer Vision*, 108(1):148–164, May 2014. `doi:10.1007/s11263-013-0675-3`. 2.2.5

[Koc90] Sandeep Kochhar. A prototype system for design automation via the browsing paradigm. In *Proceedings of Graphics Interface '90*, volume Halifax, pages 156–166, 1990. URL: `http://graphicsinterface.org/proceedings/gi1990/gi1990-19/`, `doi:10.20380/gi1990.19`. 2.2.4

[KP09] William B. Kerr and Fabio Pellacini. Toward evaluating lighting design interface paradigms for novice users. In *ACM Transactions on Graphics*, volume 28, page 1, July 2009. URL: `http://portal.acm.org/citation.cfm?doid=1531326.1531332`, `doi:10.1145/1531326.1531332`. 2.2.3, 2.2.6, 5.2, B.1, B.1.2, B.1.2

[KPC93] John K. Kawai, James S. Painter, and Michael F. Cohen. Radioptimization: Goal Based Rendering. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '93, pages 147–154, New York, NY, USA, 1993. ACM. URL: `http://doi.acm.org/10.1145/166117.166136`, `doi:10.1145/166117.166136`. 2.2.3, 5.2

[KPG15] Adriana Kovashka, Devi Parikh, and Kristen Grauman. WhittleSearch: Interactive Image Search with Relative Attribute Feedback. *International Journal of Computer Vision*, 115(2):185–210, November 2015. arXiv: 1505.04141. URL: `http://arxiv.org/abs/1505.04141`, `doi:10.1007/s11263-015-0814-0`. 2.2.2

[KSG20] Yuki Koyama, Issei Sato, and Masataka Goto. Sequential Gallery for Interactive Visual Design Optimization. *ACM Trans. Graph.*, 39(4):12, 2020. `doi:10.1145/3386569.3392444`. 2.2.5

[KSI14] Yuki Koyama, Daisuke Sakamoto, and Takeo Igarashi. Crowd-powered parameter analysis for visual design exploration. In *Proceedings of the 27th annual ACM symposium on User interface software and technology - UIST '14*, pages 65–74, Honolulu, Hawaii, USA, 2014. ACM Press. URL: `http://dl.acm.org/citation.cfm?doid=2642918.2647386`, `doi:10.1145/2642918.2647386`. 2.2.5, 3.2.5, 4.4.1

[KSSI17] Yuki Koyama, Issei Sato, Daisuke Sakamoto, and Takeo Igarashi. Sequential line search for efficient visual design optimization by crowds. *ACM Transactions on Graphics*, 36(4):1–11, July 2017. URL: `http://dl.acm.org/citation.cfm?doid=3072959.3073598`, `doi:10.1145/3072959.3073598`. 2.2.2

[L8] L8 LLC. L8. URL: `https://l8.ltd/m/lc/l8-ce.html`. 5.2

[LH14] Zhicheng Liu and Jeffrey Heer. The Effects of Interactive Latency on Exploratory Visual Analysis. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2122–2131, December 2014. URL: http://ieeexplore.ieee.org/document/6876022/, doi:10.1109/TVCG.2014.2346452. 1.1

[LMGH$^+$13] Jorge Lopez-Moreno, Elena Garces, Sunil Hadap, Erik Reinhard, and Diego Gutierrez. Multiple Light Source Estimation in a Single Image. *Computer Graphics Forum*, 2013. URL: https://diglib.eg.org:443/xmlui/handle/10.1111/v32i8pp170-182, doi:http://dx.doi.org/10.1111/cgf.12195. 5.2

[LRFH13] Sharon Lin, Daniel Ritchie, Matthew Fisher, and Pat Hanrahan. Probabilistic color-by-numbers: suggesting pattern colorizations using factor graphs. In *ACM Transactions on Graphics*, volume 32, page 1, July 2013. URL: http://dl.acm.org/citation.cfm?doid=2461912.2461988, doi:10.1145/2461912.2461988. 2.2.1, 2.2.3, 5.4.3

[LRT$^+$14] Pierre-Yves Laffont, Zhile Ren, Xiaofeng Tao, Chao Qian, and James Hays. Transient attributes for high-level understanding and editing of outdoor scenes. In *ACM Transactions on Graphics*, volume 33, pages 1–11, July 2014. URL: http://dl.acm.org/citation.cfm?doid=2601097.2601101, doi:10.1145/2601097.2601101. 2.2.2

[LS71] William H. Lawton and Edward A. Sylvestre. Self Modeling Curve Resolution. *Technometrics*, 13(3):617–633, August 1971. URL: https://www.tandfonline.com/doi/abs/10.1080/00401706.1971.10488823, doi:10.1080/00401706.1971.10488823. 2.2.2

[LSDJ06] Michael S. Lew, Nicu Sebe, Chabane Djeraba, and Ramesh Jain. Content-based multimedia information retrieval: State of the art and challenges. In *ACM Transactions on Multimedia Computing, Communications, and Applications*, volume 2, pages 1–19, February 2006. URL: http://portal.acm.org/citation.cfm?doid=1126004.1126005, doi:10.1145/1126004.1126005. 2.2.2

[LSK$^+$10] Brian Lee, Savil Srivastava, Ranjitha Kumar, Ronen Brafman, and Scott R. Klemmer. Designing with interactive example galleries. In *Proceedings of the 28th international conference on Human factors in computing systems - CHI '10*, page 2257, Atlanta, Georgia, USA, 2010. ACM Press. URL: http://portal.acm.org/citation.cfm?doid=1753326.1753667, doi:10.1145/1753326.1753667. 2.2.1

[MA] MA Lighting. grandMA3 Software. URL: https://www.malighting.com/product/grandma3-software-4022020/. 1, 2.2.6, 5.2

[Mas19] Massive Entertainment. Tom Clancy's The Division 2, March 2019. 1

[Max14] Maxis. The Sims 4, September 2014. 1

[Max18] Maxis. Spore, May 2018. 1

[MGB$^+$18] Justin Matejka, Michael Glueck, Erin Bradner, Ali Hashemi, Tovi Grossman, and George Fitzmaurice. Dream Lens: Exploration and Visualization of Large-Scale Generative Design Datasets. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, pages 369:1–369:12, New York, NY, USA, 2018. ACM. event-place: Montreal QC, Canada. URL: http://doi.acm.org/10.1145/3173574.3173943, doi:10.1145/3173574.3173943. 2.2.1

[MH08] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of machine learning research*, 9(Nov):2579–2605, 2008. 2.2.2

[MLL+15] Brad A. Myers, Ashley Lai, Tam Minh Le, YoungSeok Yoon, Andrew Faulring, and Joel Brandt. Selective Undo Support for Painting Applications. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 4227–4236, Seoul, Republic of Korea, April 2015. Association for Computing Machinery. doi:10.1145/2702123.2702543. 6.3

[MM98] Richard G. McDaniel and Brad A. Myers. Building applications using only demonstration. In *Proceedings of the 3rd international conference on Intelligent user interfaces*, IUI '98, pages 109–116, San Francisco, California, USA, January 1998. Association for Computing Machinery. doi:10.1145/268389.268409. 4.6.1

[Moz16] Mozilla. Firefox 46.0, April 2016. URL: https://www.mozilla.org/en-US/firefox/46.0/releasenotes/. 6.3

[MP09] James McCann and Nancy Pollard. Local layering. *ACM Transactions on Graphics*, 28(3):84:1–84:7, July 2009. doi:10.1145/1531326.1531390. C.2

[MPBM03] Wojciech Matusik, Hanspeter Pfister, Matt Brand, and Leonard McMillan. A Data-driven Reflectance Model. In *ACM SIGGRAPH 2003 Papers*, SIGGRAPH '03, pages 759–769, New York, NY, USA, 2003. ACM. event-place: San Diego, California. URL: http://doi.acm.org/10.1145/1201775.882343, doi:10.1145/1201775.882343. 2.2.2

[Mr.19] Mr.doob. three.js, November 2019. original-date: 2010-03-23T18:58:01Z. URL: https://github.com/mrdoob/three.js. 4.5.3, 4.6.1

[MRR+97] J. Marks, W. Ruml, K. Ryall, J. Seims, S. Shieber, B. Andalman, P. A. Beardsley, W. Freeman, S. Gibson, J. Hodgins, T. Kang, B. Mirtich, and H. Pfister. Design galleries: a general approach to setting parameters for computer graphics and animation. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques - SIGGRAPH '97*, pages 389–400. ACM Press, 1997. URL: http://portal.acm.org/citation.cfm?doid=258734.258887, doi:10.1145/258734.258887. 1.1, 2.2.1, 5.1

[MTN+15] Vittorio Megaro, Bernhard Thomaszewski, Maurizio Nitti, Otmar Hilliges, Markus Gross, and Stelian Coros. Interactive Design of 3D-printable Robotic Creatures. *ACM Trans. Graph.*, 34(6):216:1–216:9, October 2015. URL: http://doi.acm.org/10.1145/2816795.2818137, doi:10.1145/2816795.2818137. 2.2.5

[NCS16] NCSOFT. Blade & Soul, January 2016. 1

[Nev04] Neversoft. Tony Hawk's Underground 2, October 2004. 1

[Nin13] Nintendo. Mii Channel, October 2013. 1

[NN13] Donald A Norman and Donald A Norman. *The design of everyday things, revised and expanded edition*. Basic Books, New York, 2013. OCLC: 880275398. URL: http://www.books24x7.com/marc.asp?bookid=59487. 2.1.2, 2.2.6

[noa12] Move tool moves wrong layer..., May 2012. URL: https://community.adobe.com/t5/photoshop/move-tool-moves-wrong-layer/4099246?page=1. 6.3

[OLAH14] Peter O'Donovan, Jānis Lībeks, Aseem Agarwala, and Aaron Hertzmann. Exploratory font selection using crowdsourced attributes. *ACM Transactions on Graphics*, 33(4):1–9, July 2014. URL: `http://dl.acm.org/citation.cfm?doid=2601097.2601110`, `doi:10.1145/2601097.2601110`. 2.2.2

[PBMF07] Fabio Pellacini, Frank Battaglia, R. Keith Morley, and Adam Finkelstein. Lighting with Paint. *ACM Trans. Graph.*, 26(2), June 2007. URL: `http://doi.acm.org/10.1145/1243980.1243983`, `doi:10.1145/1243980.1243983`. 2.2.6, 5.2

[PD84] Thomas Porter and Tom Duff. Compositing digital images. *ACM SIGGRAPH Computer Graphics*, 18(3):253–259, January 1984. `doi:10.1145/964965.808606`. 6.3

[Pel10] Fabio Pellacini. envyLight: an interface for editing natural illumination. *ACM Transactions on Graphics (TOG)*, 29(4):34:1–34:8, July 2010. `doi:10.1145/1778765.1778771`. 5.4.4

[PF92] Pierre Poulin and Alain Fournier. Lights from Highlights and Shadows. In *Proceedings of the 1992 Symposium on Interactive 3D Graphics*, I3D '92, pages 31–38, New York, NY, USA, 1992. ACM. URL: `http://doi.acm.org/10.1145/147156.147160`, `doi:10.1145/147156.147160`. 5.2

[PG11] Devi Parikh and Kristen Grauman. Relative attributes. In *2011 International Conference on Computer Vision*, pages 503–510, November 2011. ISSN: 1550-5499. `doi:10.1109/ICCV.2011.6126281`. 2.2.2

[Pho19] Phoenix Labs. Dauntless, September 2019. 1

[Pro19] Prototypo. Prototypo, 2019. URL: `https://www.prototypo.io/`. 1, 4.5.3, 4.6.1, 4.6.2

[PTG02] Fabio Pellacini, Parag Tole, and Donald P. Greenberg. A User Interface for Interactive Cinematic Shadow Design. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, pages 563–566, New York, NY, USA, 2002. ACM. URL: `http://doi.acm.org/10.1145/566570.566617`, `doi:10.1145/566570.566617`. 5.2

[PVL+05] Fabio Pellacini, Kiril Vidimče, Aaron Lefohn, Alex Mohr, Mark Leone, and John Warren. Lpics: A Hybrid Hardware-accelerated Relighting Engine for Computer Cinematography. In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, pages 464–470, New York, NY, USA, 2005. ACM. URL: `http://doi.acm.org/10.1145/1186822.1073214`, `doi:10.1145/1186822.1073214`. 5.2, 5.5.1

[RKK11] Daniel Ritchie, Ankita Arvind Kejriwal, and Scott R. Klemmer. d.tour: style-based exploration of design example galleries. In *Proceedings of the 24th annual ACM symposium on User interface software and technology - UIST '11*, page 165, Santa Barbara, California, USA, 2011. ACM Press. URL: `http://dl.acm.org/citation.cfm?doid=2047196.2047216`, `doi:10.1145/2047196.2047216`. 2.2.1

[RRC+06] Gonzalo Ramos, George Robertson, Mary Czerwinski, Mary Czerwinski, Desney Tan, Patrick Baudisch, Ken Hinckley, Ken Hinckley, and Maneesh Agrawala. Tumble! Splat! Helping Users Access and Manipulate Occluded Content in 2D Drawings. In

Proceedings of the Working Conference on Advanced Visual Interfaces, AVI '06, pages 428–435, New York, NY, USA, 2006. ACM. event-place: Venezia, Italy. URL: `http://doi.acm.org/10.1145/1133265.1133351`, `doi:10.1145/1133265.1133351`. 2.2.6, 6.3

[RW06] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian processes for machine learning*. Adaptive computation and machine learning. MIT Press, Cambridge, Mass, 2006. OCLC: ocm61285753. 4.1, 4.1

[Sal71] Gerard Salton, editor. *The SMART retrieval system: experiments in document processing*. Prentice-Hall series in automatic computation. Prentice-Hall, Englewood Cliffs, NJ, 1971. OCLC: 214899. 2.2.4

[SCGT15] Mélina Skouras, Stelian Coros, Eitan Grinspun, and Bernhard Thomaszewski. Interactive Surface Design with Interlocking Elements. *ACM Trans. Graph.*, 34(6):224:1–224:7, October 2015. URL: `http://doi.acm.org/10.1145/2816795.2818128`, `doi:10.1145/2816795.2818128`. 2.2.5

[SDS+93] Chris Schoeneman, Julie Dorsey, Brian Smits, James Arvo, and Donald Greenberg. Painting with Light. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '93, pages 143–146, New York, NY, USA, 1993. ACM. URL: `http://doi.acm.org/10.1145/166117.166135`, `doi:10.1145/166117.166135`. 5.2

[SFPF19] Evan Shimizu, Matt Fisher, Sylvain Paris, and Kayvon Fatahalian. Finding Layers Using Hover Visualizations. *Proceedings of Graphics Interface 2019*, Kingston:28–31 May 2019, 2019. URL: `http://graphicsinterface.org/proceedings/gi2019/gi2019-16/`, `doi:10.20380/gi2019.16`. 3.2.5, 6

[SGM+16] Ana Serrano, Diego Gutierrez, Karol Myszkowski, Hans-Peter Seidel, and Belen Masia. An intuitive control space for material appearance. *ACM Transactions on Graphics*, 35(6):1–12, November 2016. URL: `http://dl.acm.org/citation.cfm?doid=2980179.2980242`, `doi:10.1145/2980179.2980242`. 2.2.2

[Shn02] Ben Shneiderman. Creativity support tools: A workshop sponsored by the National Science Foundation. *Communications of the ACM*, 45(10), October 2002. URL: `http://portal.acm.org/citation.cfm?doid=570907.570945`, `doi:10.1145/570907.570945`. 2.1.2

[Sid20] SideFX. Houdini, 2020. URL: `https://www.sidefx.com/`. 1

[Sim73] Herbert A. Simon. The structure of ill structured problems. *Artificial Intelligence*, 4(3-4):181–201, 1973. `doi:10.1016/0004-3702(73)90011-8`. 2.1.1, 2.1.2

[Sim08] Herbert Alexander Simon. *The sciences of the artificial*. MIT Press, Cambridge, Mass., 3. ed., [nachdr.] edition, 2008. OCLC: 552080160. 2.1.2

[SMD+15] Leonid Sigal, Moshe Mahler, Spencer Diaz, Kyna McIntosh, Elizabeth Carter, Timothy Richards, and Jessica Hodgins. A perceptual control space for garment simulation. *ACM Transactions on Graphics*, 34(4):117:1–117:10, July 2015. URL: `http://dl.acm.org/citation.cfm?doid=2809654.2766971`, `doi:10.1145/2766971`. 2.2.2

[Sof] CAST Software. wysiwyg Lighting Design. URL: `https://cast-soft.com/`

wysiwyg-lighting-design/. 5.2

[SP] Sara L Su and Sylvain Paris. QuickSelect: history-based selection expansion. page 7. 6.3

[SPF+19] Evan Shimizu, Sylvain Paris, Matt Fisher, Ersin Yumer, and Kayvon Fatahalian. Exploratory Stage Lighting Design using Visual Objectives. *Computer Graphics Forum*, 38(2):417–429, 2019. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13648. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13648, doi:10.1111/cgf.13648. 5

[SQRH+16] Stephan Streuber, M. Alejandra Quiros-Ramirez, Matthew Q. Hill, Carina A. Hahn, Silvia Zuffi, Alice O'Toole, and Michael J. Black. Body talk: crowdshaping realistic 3D avatars with words. *ACM Transactions on Graphics*, 35(4):1–14, July 2016. URL: http://dl.acm.org/citation.cfm?doid=2897824.2925981, doi:10.1145/2897824.2925981. 2.2.2

[Squ20] Square Enix. Final Fantasy XIV, February 2020. 1

[SSCO09] L. Shapira, A. Shamir, and D. Cohen-Or. Image Appearance Exploration by Model-Based Navigation. *Computer Graphics Forum*, 28(2):629–638, April 2009. URL: http://doi.wiley.com/10.1111/j.1467-8659.2009.01403.x, doi:10.1111/j.1467-8659.2009.01403.x. 2.2.3

[Sub19a] Substance3D. Substance Designer, January 2019. URL: https://www..substance3d.com. 1, 2.2.2, 4.6.1

[Sub19b] Substance3D. Substance Source, 2019. URL: https://source.substance3d.com. 7.6

[SW14] Michael Schwarz and Peter Wonka. Procedural Design of Exterior Lighting for Buildings with Complex Constraints. *ACM Trans. Graph.*, 33(5):166:1–166:16, September 2014. URL: http://doi.acm.org/10.1145/2629573, doi:10.1145/2629573. 2.2.3, 5.2

[SYS20] SYSTEMAX Software Development. PaintTool SAI, 2020. URL: https://www.systemax.jp/en/sai/. 6.3

[Tec19] Unity Technologies. Unity3D, 2019. URL: https://unity3d.com/. 1

[TGY+09] Jerry O. Talton, Daniel Gibson, Lingfeng Yang, Pat Hanrahan, and Vladlen Koltun. Exploratory modeling with collaborative design spaces. *ACM Transactions on Graphics*, 28(5):1, December 2009. URL: http://portal.acm.org/citation.cfm?doid=1618452.1618513, doi:10.1145/1618452.1618513. 2.2.1

[The20] The GIMP Team. GIMP, 2020. URL: https://www.gimp.org/. (document), 6.3, 6.2

[TM02] Michael Terry and Elizabeth D. Mynatt. Side views: persistent, on-demand previews for open-ended tasks. In *Proceedings of the 15th annual ACM symposium on User interface software and technology - UIST '02*, page 71, Paris, France, 2002. ACM Press. URL: http://portal.acm.org/citation.cfm?doid=571985.571996, doi:10.

1145/571985.571996. 2.2.6, 3.2.5, 4.4.2

[/u/11] /u/mystline7. If you use Photoshop and don't name your layers I hate you., January 2011. URL: https://www.reddit.com/r/web_design/comments/ev4nf/if_you_use_photoshop_and_dont_name_your_layers_i/. 6

[/u/19] /u/PiptheMuffin. The Photoshop struggle, July 2019. URL: https://www.reddit.com/r/memes/comments/cib9op/the_photoshop_struggle/. 6

[UIM12] Nobuyuki Umetani, Takeo Igarashi, and Niloy J. Mitra. Guided Exploration of Physically Valid Shapes for Furniture Design. *ACM Trans. Graph.*, 31(4):86:1–86:11, July 2012. URL: http://doi.acm.org/10.1145/2185520.2185582, doi:10.1145/2185520.2185582. 2.2.5

[Vec20] Vectorworks, Inc. Vectorworks Spotlight, 2020. URL: https://www.vectorworks.net/en-US/spotlight. 5.2

[WKB+16] Andrew M. Webb, Andruid Kerne, Zach Brown, Jun-Hyun Kim, and Elizabeth Kellogg. LayerFish: Bimanual Layering with a Fisheye In-Place. In *Proceedings of the 2016 ACM International Conference on Interactive Surfaces and Spaces*, ISS '16, pages 189–198, Niagara Falls, Ontario, Canada, November 2016. Association for Computing Machinery. doi:10.1145/2992154.2992171. 2.2.6, 6.3

[Wol19] WolfSkullJack. WolfSkullJack on Twitter: "Merge the layer, Kronk WROOONGGG LAYYYYYERRR" / Twitter, July 2019. URL: https://twitter.com/wolfskulljack/status/1149251320597155840. 6

[WQM+17] Kanit Wongsuphasawat, Zening Qu, Dominik Moritz, Riley Chang, Felix Ouk, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer. Voyager 2: Augmenting Visual Analysis with Partial View Specifications. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems - CHI '17*, pages 2648–2659, Denver, Colorado, USA, 2017. ACM Press. URL: http://dl.acm.org/citation.cfm?doid=3025453.3025768, doi:10.1145/3025453.3025768. 2.2.4

[YCHK15] Mehmet Ersin Yumer, Siddhartha Chaudhuri, Jessica K. Hodgins, and Levent Burak Kara. Semantic shape editing using deformation handles. *ACM Transactions on Graphics*, 34(4):86:1–86:12, July 2015. URL: http://dl.acm.org/citation.cfm?doid=2809654.2766908, doi:10.1145/2766908. 2.2.2

[Yuk19] Yuke's. WWE 2K19, October 2019. 1

[YYT+11] Lap-Fai Yu, Sai-Kit Yeung, Chi-Keung Tang, Demetri Terzopoulos, Tony F. Chan, and Stanley J. Osher. Make it home: automatic optimization of furniture arrangement. *ACM Transactions on Graphics*, 30(4):1, July 2011. URL: http://portal.acm.org/citation.cfm?doid=2010324.1964981, doi:10.1145/2010324.1964981. 2.2.3

[Zen14] Zenimax Online Studios. The Elders Scrolls Online, April 2014. 1