iPlot: An Intelligent Lighting Design Assistant

by

Andrew J Perelson

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2005

Author . . ..
Department of Electrical Engineering and Computer Science
May 6, 2005

Certified by . . ..
Kimberle Koile
Research Scientist

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

iPlot: An Intelligent Lighting Design Assistant

by

Andrew J Perelson

Submitted to the Department of Electrical Engineering and Computer Science
on May 6, 2005, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

iPlot is an intelligent lighting design assistant. Given a performance space and a set of lighting goals, each specifying an area to be lit and a direction, iPlot explores the space of possible light arrangements in search of solutions that satisfy the goals. It employs a generate, test, and repair strategy in which solutions are generated and tested to see if they satisfy a goal. If a goal failed to be satisfied iPlot uses the explanation of this failure to propose a number of repair suggestions that either modify the solutions or relax the set of goals to form a new goal set. It then carries out these suggestions to create new solutions viewable by the designer as a light plot, a two-dimensional top down view of the performance space, lighting pipes, and lights all drawn to scale. This thesis describes iPlot and an experiment that involved asking a lighting designer to evaluate the light plots that iPlot produced.

Thesis Supervisor: Kimberle Koile
Title: Research Scientist

Acknowledgments

No person works entirely on their own, and I would like to thank all of the people who helped me along the way to completing this thesis.

Kimberle Koile for being an expert in everything I ever need to know about intelligent design tools, search, and anything AI in general. Not to mention a friend and extremely efficient and effective editor.

Buddhika, Max, and Adam for always being around in lab when I needed help or just someone to run an idea by, complain to, or relax with. You helped me stay on track and enjoy my work.

Kevin Quigley and Stephen Peters, who ruthlessly attacked any systems problems I had with my computers while working and helping me install numerous packages on our lab's computers.

Amy Mueller for long jogging sessions without actually going anywhere, more thoughts on computer science and theater, and most importantly–keeping me sane.

Jon Wolfe and for long walks through Cambridge talking about lighting design, computer science, and just how one might manage to put them together.

Everyone in the Theater Arts department at MIT especially Mike, Karen, Janet, and Michael for all of their support and guidance with all things theater and quite a few things not theater.

Megan and my family for supporting me throughout, listening to me talk about projects they probably don't care about, and still being willing to proof read my thesis at the end of it.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

iPlot is a system designed to aid theatrical lighting designers in creating a physical representation of their lighting design concept. This physical representation is a light plot, which is a two-dimensional top-down view of the theater showing the stage, lighting pipes, lights, and circuits into which each light is plugged. iPlot explores different mappings from a designer's lighting goals to a physical model of lights, along with their positions and focus points. It evaluates design goals, proposes and refines repair suggestions for unsatisfied goals, and carries out those suggestions to create revised goals and new solutions. As such, iPlot assists designers in specifying lighting goals and exploring the space of possible solutions, or physical representations, of those goals.

1.1 Understanding Theatrical Lighting Design

Theatrical lighting design is the process of determining the placement, focus and use of lights throughout a theatrical performance. In creating a lighting design, a designer is primarily concerned with supporting the theatrical production; by finding a way for lighting to emphasize important aspects of the performance. As Stanley McCandless said "[stage] lighting may be defined as the use of light to create a sense of visibility, naturalism, composition and mood, (or atmosphere)" [9]. This lighting design process breaks down into four steps: creating an abstract design concept,

turning that abstract concept into a set of lighting goals, meeting those goals by finding a set of lights and their corresponding hang positions and focus points, and deciding how to use those lights during each moment of a performance.

The abstract design concept is the set of impressions the designer wants to convey to the audience using lighting; the impressions can be thought of as "looks" or moods and are usually described in terms of familiar styles, locations, objects, and adjectives. A lighting designer, for example, might describe his abstract goals by saying the lighting should be "film noir", "like a creepy 1950's diner in a scary movie", "angry", or "like a normal, everyday room". These goals specify the lighting in terms of a scene or mood that people can imagine.

From the abstract design concept the designer must articulate lighting goals that, when realized, will create a look that matches the abstract design concept. These lighting goals describe what section of the stage is illuminated and from what direction. Examples of lighting goals are "front light coming at 45 degrees above the actors, covering the entire stage" or "a single pool of light dead center, about half the size of the stage, coming from directly above". Thus to go from abstract design concept to physical lighting goals the designer must devise methods that enable lighting to imply a given situation or circumstance. How can lighting make a room look scary, for example? A possible method is to use front light coming from the feet of the people on stage, i.e., light that illuminates below a person's face from below. (People even use this method when telling a scary story—they shine a flashlight up at their face from below.)

Given a physical lighting goal, such as the ones mentioned above, and a specific theater, a designer then must determine where to hang lights and at what point on stage to focus them. He uses his knowledge of the photometric properties of the lights and a fair amount of geometry and trigonometry to accomplish this task. The result is an arrangement of lights described by a light plot, a two dimensional top-down view of the theater showing the stage, the lighting pipes, lights themselves, and circuits into which each light plugged.

After placing all the lights in the theater, the designer's last task is to decide for

each moment of the show what lights should be on and how bright they should be.

1.2 Problem

iPlot is focused on helping designers with the third step in the lighting design process described above—translating physical lighting goals into a light plot that satisfies those goals. iPlot, in other words, takes a list of goals such as the following:

1. Full stage wash coming from downstage at an angle of 45°
2. Full stage wash coming from stage left at an angle of 60°
3. Full stage wash coming from stage right at an angle of 60°
4. A square 3' on a side centered on center coming from upstage at an angle of 75° .

and finds a set of lights that satisfies those goals. The set of lights is represented by a light plot, e.g. Figure 1-1.

1.3 Scenario

As a demonstration of how we envision iPlot being used, consider the following scenario in which a lighting designer, Charles, works with his lighting design assistant, iPlot.

Charles is working on a new show for the Wang Theater. He has received the architectural plans, including lighting pipe layout, and lighting inventory from the building manager. After reviewing the script, seeing a rehearsal, and meeting with the director, Charles has determined an abstract design concept. In this case, he wants to use lighting to create an environment on stage that is very film noir and creepy.

Charles next uses an intelligent assistant iConcept to determine possible physical lighting goals that would satisfy these abstract goals. These goals will be stated in

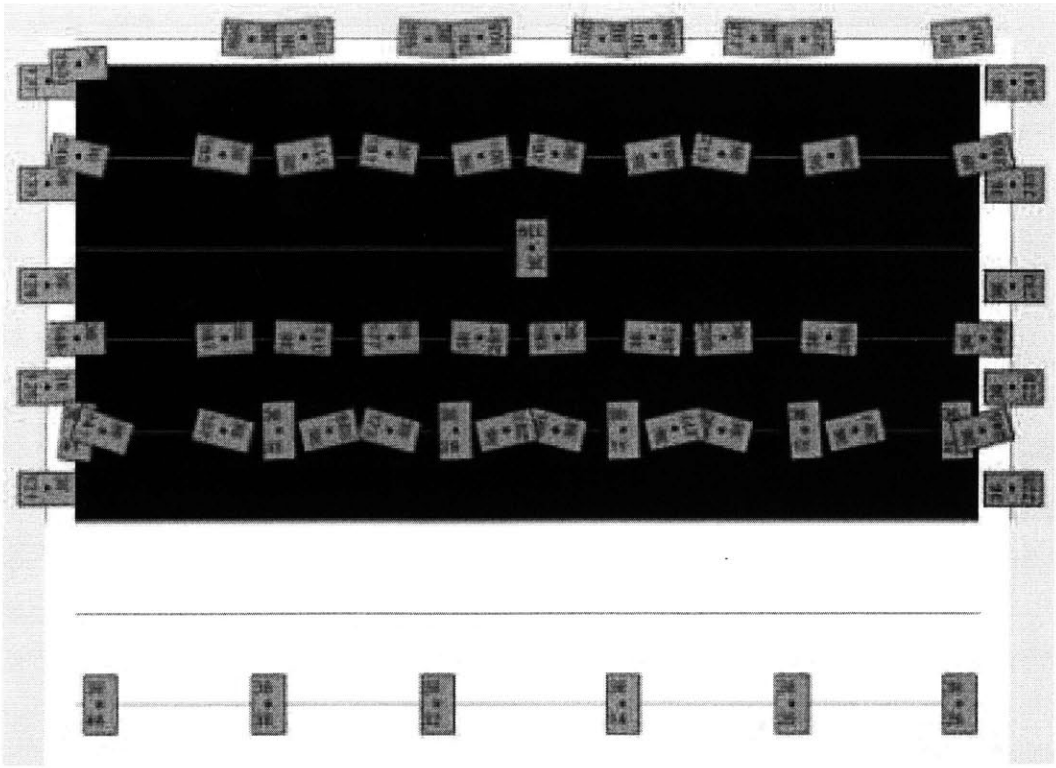


Figure 1-1: Light plot output from iPlot. This light plot shows iPlot's proposed solution to the four goals stated in Section 1.2.

terms of the area of the stage to be illuminated and from what direction. Given a representation of the production space and the abstract design concept describing the desired look, iConcept suggests using two groups of lights, which it represents as two physical lighting goals. These lighting goals specify that the first group of lights should illuminate the entire stage, with the light coming from downstage right. The second group of lights should illuminate center stage in a circle whose diameter is approximately one third that of the stage.

Charles' next task is to create a light plot. With the set of lighting goals proposed by iConcept in hand, Charles starts iPlot, loads in the information on the Wang Theater, and asks iPlot to propose a light plot. iPlot retrieves the lighting goals from iConcept. iPlot suggests a group of fifteen lights to accomplish the first goal and a single light to accomplish the second. It then renders a picture of what the Wang Theater would look like with those sixteen lights on. Charles likes the look of the

stage, but wants to modify it slightly and explore some alternatives. He drags light icons around on the screen to simulate new lights and positions, and iPlot change the look of the stage accordingly. When he is satisfied with the way the stage will look, Charles saves his work, sends a copy of the design to the director, prints out a light plot, and goes to discuss the design with the director.

1.4 Motivation

iPlot supports the early stage of lighting design with the aim of helping designers quickly find an arrangement of lights that satisfies their design concept and lighting goals. We choose to focus on this step in the lighting design process because it is the one least covered by current lighting design tools, and it is sufficiently complex that artificial intelligence techniques have the potential to provide valuable assistance to human designers (e.g. [7]).

The process of going from physical lighting goals to a light plot is one of the most time consuming and tedious tasks for a designer. It involves an extensive search using trigonometry, geometry, and lighting design knowledge to map from the goal space (stage area and light direction) to the solution space (lights with hang and focus points–a light plot). Most lighting designers are not adept with the necessary math and use rough approximations when creating plots. Current tools help with some of the math, but none attempt to generate an entire solution given a set of lighting goals. The system described in this thesis acts as an intelligent design assistant, translating a designer's lighting goals into a recommended light plot, which represents the physical layout of the lights. With a tool such as iPlot designers can explore many more design possibilities easily and find a solution more quickly than is possible with currently available tools.

Design is an inherently knowledge-based, intelligent behavior. A designer has a number of goals that he wants to accomplish and must figure out how to build something that accomplishes them. The design process itself is not static and the goals of the design often change while searching for a solution. In lighting design,

the quality of a solution to a goal is somewhat subjective. Designers have preferences that influence the solutions they create and prefer. Because of these complications, building computational tools to aid the design process requires addressing interesting AI issues such as how can the search space for design solutions be restricted to avoid the combinatoric problems inherent in large search spaces. How can one find solutions for multiple conflicting goals? This thesis proposes answers to these questions.

1.5 iPlot's Intelligence

iPlot's intelligence derives from it's knowledge base and its informed search of both solution and repair space, an idea explored in [7].

iPlot's knowledge consists of knowledge about spatial reasoning and knowledge about the lighting domain. The domain knowledge includes an awareness of the objects relevant to design (i.e., lights, performance spaces, etc.), the photometric properties of lights, and general techniques for placing lights, evaluating proposed solutions, and modifying both goals and groups of lights that form a solution to that goal. Much of iPlot's intelligence derives from being able to map not only light goals to light arrangements, but desired changes in light arrangements to changes in goals.

Exhaustive search of the design space is not a feasible method of finding an arrangement of lights for a set of goals. Not only is it computationally difficult, but many goal sets are over-constrained such that no solution is possible. iPlot is able to limit this search by using a design process similar to that of a lighting designer, with techniques for generating individual groups of lights that are best able to satisfy each goal, modifying solutions, and relaxing goals as necessary to get an acceptable solution.

1.6 Guide to this document

This document describes the iPlot system, what it accomplishes, how it works, and why it is a useful tool.

- Chapter 1 introduced the problem, the context of the problem, and iPlot's solution and discusses the motivation for building iPlot
- Chapter 2 describes a lighting designer's design process and how iPlot models it
- Chapter 3 describes iPlot's domain knowledge and representation
- Chapter 4 describes iPlot's control structure
- Chapter 5 describes iPlot's user interface
- Chapter 6 describes an evaluation of iPlot given a realistic design problem
- Chapter 7 discusses related work in artificial intelligence and lighting design tools
- Chapter 8 summarizes iPlot's contributions and discusses future work.

Chapter 2

Approach

In building iPlot we adopt the view that the best lighting design tool serves as an assistant modeled on the way human lighting designers create light plots[1]. In order to aid Charles with his design process, as in the scenario from Chapter 1, the system has to be capable of representing Charles' lighting goals and mapping them to a physical model of light positions and focus points. In particular, it must find a set of available lights that can accomplish the specified goals when placed in the theater's light hanging positions. It also must display its recommendation in such a way that it is understandable to Charles and easily explored and modified. The proposed system, iPlot, is capable of performing each of these tasks.

Given a set of physical lighting goals, determining which lights to use for each goal and where to put them is an over-constrained problem. A designer always can specify more goals than any given set of lights can satisfy. Consider a trivial case where three lights are available and the lighting designer wants four distinct, non-overlapping pools of light on stage, each from a different direction. Even such a small scale problem has no solution. A designer would be forced to abandon that set of goals or rent more lights. The problem also can be over-constrained when determining a light's physical placement. If a room has only one hanging position for a light, then only a single direction is possible for any given point on stage regardless of the desired direction. While the situation is not always this bad, designers often must place lights

[1]This view is similar to that adopted in [7]

in less-than-optimal positions that are as close as physically possible to where they would really like them to be.

For a realistic size theatrical production in which it is possible to realize all of the lighting goals, the search space that must be explored to find a solution is huge. Imagine, for example, a situation in which there are 15 different goals and 100 lights. Each light can be assigned to one of 15 goals, independent of every other light. Trying every possible combination would result in looking at 15^{100} combinations of lights, each of which would be a potential solution and thus would need to be evaluated.

In spite of a very large search space, lighting designers manage to create light plots with a very limited amount of search. They accomplish this task by having a very efficient set of preferences that effectively prune the search tree. This preferential, or informed, search on the part of a lighting designer amounts to having a specific style of design, which guides their design choices, and a large case base of previous design experience to give them examples of what can be accomplished with a given inventory of lights. Similarly, iPlot can use informed search to quickly generate a set of light placements that satisfy the goals of a lighting designer. In creating a design methodology for iPlot, it is useful to first examine in a more detail a lighting designer's approach to the problem.

2.1 A lighting designers approach

Let us again consider the scenario presented earlier. If a designer did not use a system such as iPlot, then he would have to create a light plot from his lighting goals by hand. A simplified overview of a lighting designer's process in this case might be something along the following lines. After creating a list of physical goals the lights used should satisfy the lighting designer sits down to work out exactly how he should place the lights in the theater. The designer will step through the goals one by one, and for each one, create a group of lights that satisfies that goal. Then, for each group of lights, the designer will estimate the number of lights needed for that group, and if there are not enough lights, he will go back and either decrease the area of the stage

covered by a lighting goal or remove a goal altogether, thus reducing the number of lights needed to satisfy all the goals. For each group of lights a designer thinks will work for a specific group, the designer then explores the placement of lights within that group, checking for alternative arrangements that may reduce the number of lights.

Initially this process of placing lights involves merely working through the trigonometry to find the best place for a light and the correct spacing between lights. As more and more lights are in the space, however, conflicts may arise between two lights that need to be in the same place. Often the designer will move one light over slightly so that the two lights are centered around their desired location. Repeating this action many times may result in locations quite far from the desired location, however, and the goal will not be satisfied. At this point the designer will explore either alternative goals that can be accomplished more easily, or looks at ways to create more hanging positions at the desired location.

It is important to note that in the description above the lighting designer assumed goal independence, finding a separate set of lights to satisfy each goal. Discussing the problem with lighting designers, we have found this assumption to hold because they are internally combining all dependent goals into a single goal. They are thinking about the goals whose solutions will be the most flexible, i.e. able to satisfy the largest set of dependent goals. As an example, consider that a designer might describe a goal stating that he wants the entire stage to be illuminated and the focus points of the lights to be symmetric about center with a column of focus points to be on center stage. This goal covers a single area, but has additional constraints on how individual lights are used to satisfy this goal. Questioning the designer you would find that he wanted to use the solution of this goal to:

- Illuminate the entire stage
- Illuminate stage right
- Illuminate stage left

- Illuminate a column several feet wide, centered on center stage

While the designer conceptually thought in terms of a single goal whose solution would have many applications, the goal could also be decomposed into four dependent goals. Thus, in effect the designer reduced four dependent goals into one goal that is further constrained. As a result, when the designer sought a group of lights that satisfied each goal, he was considering the goals independent because the dependent goals had already been combined.

2.2 iPlot's approach

To accomplish its task of mapping physical lighting goals to a physical model of light positions, iPlot models a human designer's search process. Given a set of goals and preferred methods for satisfying each goal, it first assumes goal independence, and for each goal attempts to create a solution, which is represented by a set of lights, their hanging positions, and their focus points on the stage. If the system fails to find a solution (because a needed light or position is unavailable), it attempts to find an acceptable solution by moving lights or modifying goals. Thus, iPlot is using a generate, test, and repair paradigm to find a solution. It first generates a potential solution, and then tests that solution. If the solution is not satisfactory, it uses the results of the test to modify its goals and generate a new solution. Unlike most generate, test, and repair systems (e.g. [7]), iPlot modifies goals as well as solutions.

To accomplish this task, iPlot employs lighting design and spatial reasoning knowledge. The lighting design knowledge can be garnered from lighting designers, light specifications and the relationship between those specifications. For example, light manufacturers publish the specifications of their lights, including photometric information such as the beam and field angles, the intensity of the light, and how fast the intensity decreases as you move out from the center of the light. Using this photometric information, a set of light placements, and a model of the performance space, iPlot can predict what the stage will look like when those lights are turned on.

Conceptually iPlot consists of four components: (1) a physical model of the space

for which a designer is creating a light plot; (2) a base of knowledge used to place lights, i.e. when and how to calculate the direction of a light beam and predict what a light will look like on stage; (3) a way of representing lighting goals –including an understanding of how they can be satisfied, fail to be satisfied, and ways of repairing goals that fail to be satisfied; (4) a search system that will attempt to match lights and placements to goals; Parts one through three of this system form the domain knowledge needed to address the problem, while part four is the control structure that manipulates this knowledge to find a solution. The relationship between these four components is shown in Figure 2-1

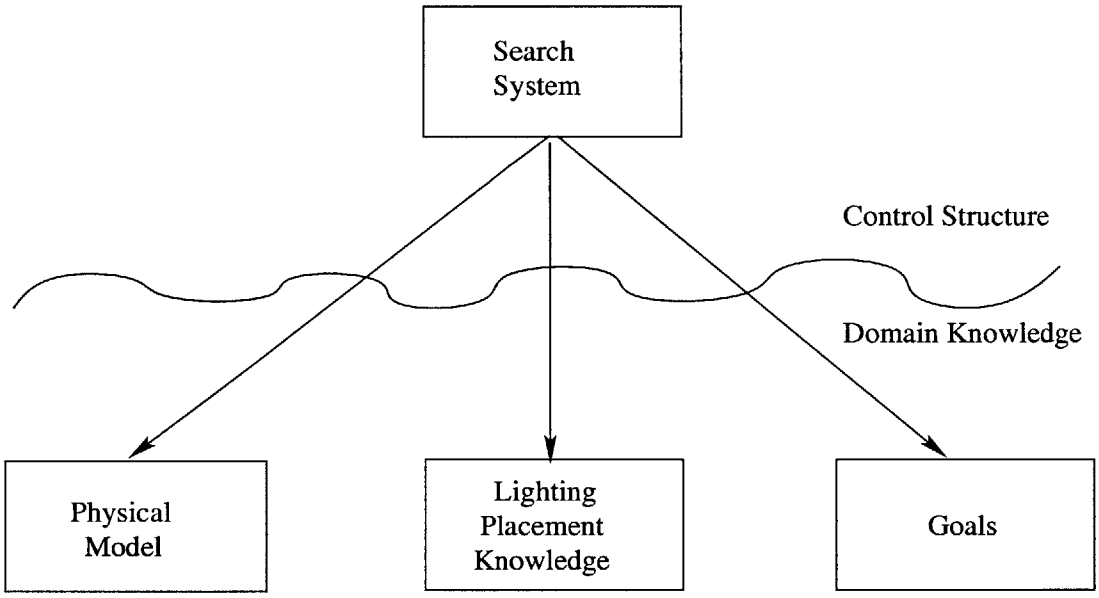


Figure 2-1: iPlot System Overview. The search system manipulates a physical model using lighting placement knowledge to find a solution to a set of goals.

Chapter 3

Domain Knowledge & Representation

To map from lighting goals to physical light placements iPlot needs an internal representation of the objects and knowledge being used to create a solution. This representation must include knowledge about performance spaces and all of the items within it such as lights, pipes on which lights are hung, and the stage lights are focused on; a representation of the goals that it is attempting to satisfy and the requirements that have to be met for a group of lights to be considered a solution to a given goal; and ways of refining goals for which no solution can be generated. In addition to this, iPlot requires knowledge of how the objects being represented can be manipulated and how they interact with each other. In iPlot, this knowledge of the properties, behavior, and interaction of objects is contained within the representation of the objects themselves.

3.1 Performance Space

iPlot creates a representation of the physical objects that are in a theater, or any performance space, that are relevant to lighting design. This initial version of iPlot uses a simple model of the performance space and considers only the stage, the pipes on which lights can be hung, and the individual lights that are hung. In addition to

these physical objects, we also represent the inventory, which is the collection of lights available for use in a specific theater. For a description of the geometric constructs used to build the spatial representation of these objects see Appendix A.

The control structure must go through the performance space to access any object within the space. Thus to hang a light the control structure gives a light and a hang point to the performance space. The performance space hangs the light if there is an available hanging position at that point.

3.1.1 Stage & Coordinate Space

The stage in iPlot is represented as a two-dimensional rectangle with a constant z-coordinate of zero and serves as a reference for designers when talking about directions. All directions are given from the perspective of someone on stage, facing the audience. Designers also use the downstage center point on the stage as the origin of the coordinate space when talking about the position of any object in the theater, accordingly the origin of iPlot's coordinate space is always the downstage center point of the stage. As shown in Figure 3-1, the x-axis runs along the bottom of the stage, the positive direction going stage left and the y-axis runs along the centerline of the stage, the positive direction going upstage. The z-axis is perpendicular to the stage, the positive direction is towards the ceiling.

The iPlot coordinate space is discrete, taking on only integer values in which one unit represents one inch. This discretized coordinate space was chosen to make the computational geometry algorithms necessary to reason about the space simpler and easier to formulate. It is assumed that having the smallest unit be an inch will not hinder the performance of the system in determining light placements. This assumption is reasonable because when placing lights in the theater, the hung position of a light is always approximated at most to the closest inch. Furthermore, given that the audience is far from the stage, the difference of an inch in placement or focus point rarely, if ever, changes a human's perception of a light beam.

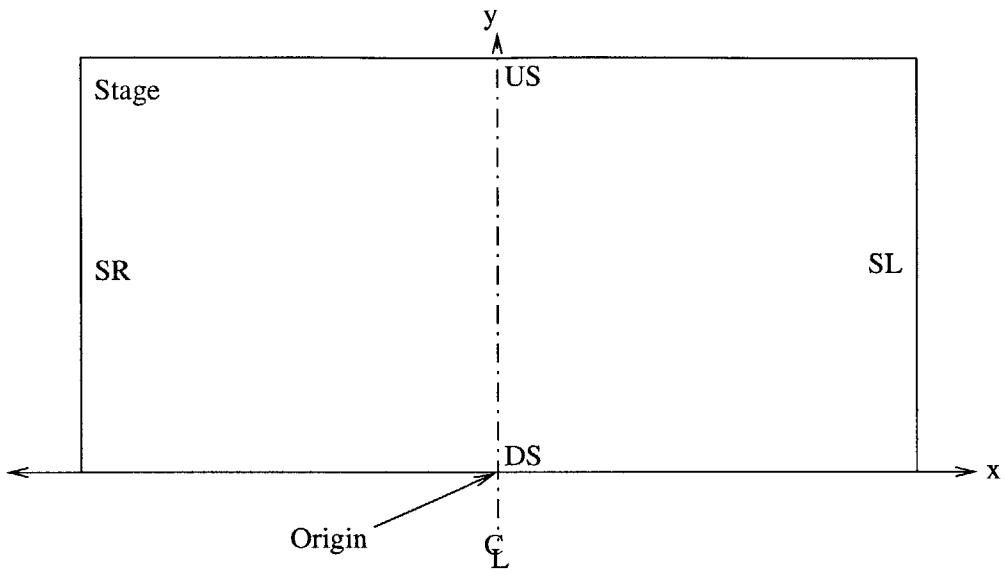


Figure 3-1: Stage & Coordinate Space. The origin of the coordinate space is at the most downstage center point on the stage, with the x-axis on the downstage edge of the stage and the y-axis on the centerline

3.1.2 Pipes

Pipes in iPlot represent the structural pipes in the theater that lights can be attached to. In the theater the standard lighting pipe is the 1.5 inch inner-diameter schedule forty pipe. The system makes the simplifying assumption that all pipes are straight lines and furthermore the connection points between pipes and the buildings superstructure are not modeled.

The length of each pipe is subdivided into three types of areas: available areas that are unoccupied by a light; forcible areas that are occupied by a light that can move and still satisfy its goal; and unavailable areas that are occupied by lights that if moved will no longer satisfy their goals. In creating this set of three areas Pipes are capable of reasoning about lights in relation to the bounded direction of their goal. A light is said to "satisfy its goal" with its placement on the pipe if the resulting line between source point and focus point is within the bounds of the goal direction when centered on the focus point. Thus a pipe can distinguish unavailable positions from forcible ones by testing to see if a light can be moved to the adjacent possible hang

point and still satisfy its goals.

Given a light, and a hang point on a pipe, the pipe will hang the light on that point, if all points in space the light would occupy in that hang position are currently available or forcible.

3.1.3 Lights

Light Categories & Types

A wide variety of theatrical or entertainment lighting fixtures are commercially available. While the exact specifications vary between lights, the lights can be grouped into a few categories according to the lens or reflector of the light. Currently iPlot's knowledge of light categories encapsulates three of the most popular conventional (non-moving) light types: ellipsoidal reflector spotlight (ERS), Fresnel spotlight (Frensel is a lens type named after its inventor Augustine Fresnel) , and the parabolic aluminum reflector spotlight (PAR).

Every light in iPlot has a type, which is the conventional name for that light and in general includes information on the manufacturer and the properties of the light. Each light type belongs to one of the light categories. Given a specific light type, iPlot can determine to which category of lights it belongs. Furthermore, given a category of lights, iPlot can enumerate the different light types it supports that are in that category.

Light Properties

In reasoning about where to place lights to achieve a given lighting look, or state of illumination, we want our representation of lights to encapsulate the physical properties relevant to placing a light, and the photometric properties of the light.

The system approximates the physical form of a light using a bounding cube. The placement and focus of a light is determined by using two points: the hang point, where a light attaches to a pipe; and the focus point, the point is space where the center of the light beam is pointed. The light beam emitted from a light is a

cone whose angle is specified by the manufacturer as a beam spread. For a complete description of how the manufacturer's specify the beam spread see Appendix B

in iPlot a light object can calculate its illumination at any point in the performance space in absolute terms, measured in foot-candles, or as a percentage of its illumination at the center of the light beam[1]. As shown in later sections, this is used by the control structure when determining how to overlap to lights being used to satisfy a single goal.

3.2 Goals

In addition to representing the physical objects that must be manipulated to satisfy a goal, the system must represent the goals themselves. A goal and its accompanying solution have five components: a purpose; a desired outcome specified as an area to illuminate and a direction; user-specified preferences that narrow the search space; whether or not the goal was satisfied, a solution if one was found. The the goal failed, it contains an explanation of which constraint was violated and how.

The purpose of a goal represents the design rationale, a brief explanation of why the designer created the goal. As discussed earlier, the desired outcome is specified as an area and a direction. In iPlot the area is represented as a rectangle and the direction as a direction vector[2]. For a complete description of these geometric constructs see Appendix A.

Each goal object contains a flag indicating whether or not it was satisfied. When a goal is satisfied, its solution is a set of lights hung in the performance space that together illuminate the goal area. When the goal failed, the goal object generates an explanation for the designer as to why the goal failed, what hard constraints were violated, and how.

[1]How iPlot calculates this intensity is shown in Appendix B

[2]Direction objects in iPlot are bounded. A designer gives bounds for the direction with the understanding that given the relative sparseness of hanging positions with a space that an exact direction for the light is not always possible. Using bounds a designer can represent the idea of an acceptable range of directions around the ideal one

3.2.1 Goal Preferences

The user's goal preferences narrow the search space by functioning as a set of soft constraints,i.e. ones that can be violated if necessary, on the generators used to create a solution. These preferences influence the overall style of the look created by a group of lights and directly or indirectly specify the type of lights used to satisfy a goal, the uniformity of illumination in the goal area, the search technique to find the solution, and the importance of a single goal in relation to the other goals of the designer. We will briefly go over the most influential preferences; a full description can be found in Appendix C.

The type of light used to satisfy a goal is specified either directly, by choosing a light type, or indirectly using a light category and a discretization, indicating how many lights should be used to span the width of the goal area.

The uniformity of illumination in the goal area is determined by two preferences the crossover and border intensity. These intensities are specified as percentages of the hotspot intensity, which is the brightest spot in the light beam[3]. The crossover intensity indirectly specifies the point at which two light beams should overlap. Similarly, the border intensity specifies how bright the border must be. Figure 3-2 shows the crossover and border intensity ellipses from a light beam. These ellipses bound the portion of the light beam that is at least as bright as the specified percentage of the hotspot. From this figure we can see that for a border to be "sufficiently" illuminated, i.e., at least as bright as the border intensity with respect to the hotspot, then it must intersect the light's border intensity ellipse. Similarly two lights that want to overlap such that two points at the crossover intensity overlap, then the crossover intensity ellipses of the two lights must just overlap each other.

The search technique, i.e., how and in what the order lights are placed, is determined by a generation start point and a generation technique. These two preferences uniquely identify one of the methods available for generating a solution to a goal.

The relative importance of a goal is determined by its order, i.e., order determines a goal's priority in being satisfied relative to other goals. This goal order is used in

[3]The hotspot is usually at the center of the light beam, which also contains the focus point.

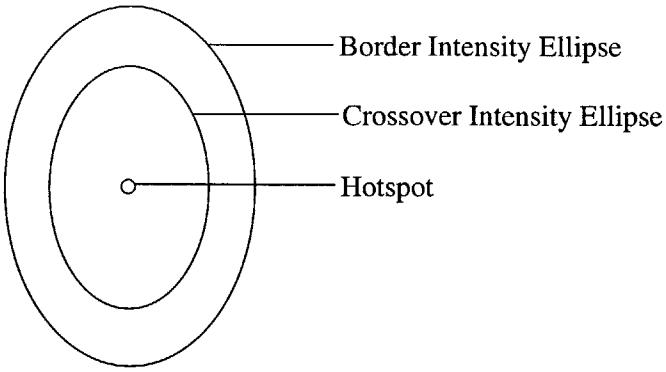


Figure 3-2: Light Beam Intensity Ellipses. We see here the ellipse formed by the intersection of the light beam (a cone) with the goal area. The intensity of the light is always brightest at the center of the beam, also called the hot spot, and fades as the distance from the center increases. The crossover intensity ellipse forms the bounds of the portion of the light beam that is at least as bright as the specified crossover intensity (which is a percentage of the hot spot intensity). Similarly, the border intensity ellipse bounds the portion of the light beam that is at least as bright as the specified border intensity.

determining the order in which goals are generated and what repair suggestions are carried out to repair a failed goal.

3.2.2 Solution Constraints

In order to satisfy a goal any group of lights must satisfy two constraints: it must cover the area specified by the designer; and each light used must come from the specified direction, measured from the light's focus point. iPlot is able to check these hard constraints for each proposed solution that it generates.

These hard constraints can fail to be met by a proposed solution in one of three ways: there may not be enough lights to cover the goal area, there may be no place to hang a light, or the desired direction may not be achievable with the available light hanging positions. These three general reasons for failure can be caused by several events, as shown in Figure 3-3. In the case that the desired direction is not achievable, this state is further decomposed into three cases: the desired xy-angle is not achievable; the desired z-angle is not achievable; or, both angles are not achievable.

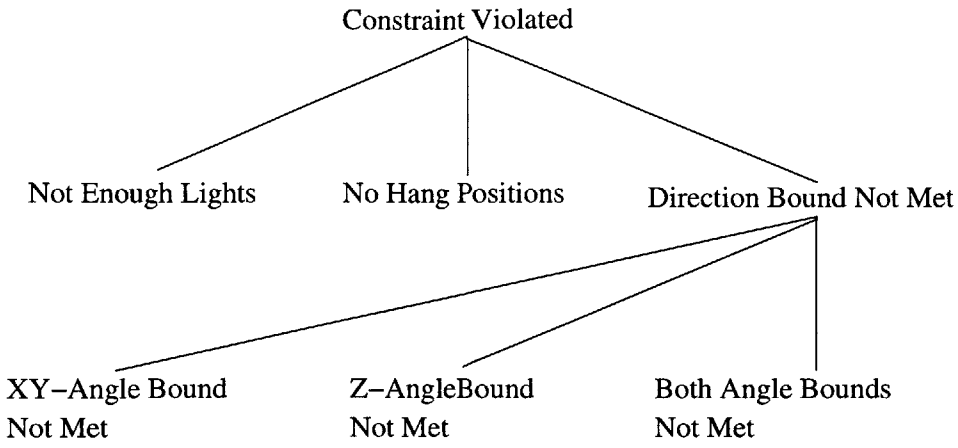


Figure 3-3: Reasons for goal failure. At the highest level a goal fails either because there are not enough lights, there are no hanging positions available, or the goal direction is not achievable. These reasons can be further decomposed. The leaves of this tree enumerate the specific reasons that a goal can fail, i.e., exactly what prevented the hard constraints from being satisfied.

3.3 Repair Suggestions

This section describes the types of repair suggestions. Their use in the solution generation and repair process is discussed in Section 4.2.

When a goal cannot be satisfied the system can create a set of repair suggestions. These repair suggestions fall into three broad categories: altering solutions, relaxing goals, and removing goals. When followed, these repair suggestions change a goal or solution to form a better proposed solution for the failed goal–one that does not violate the hard constraint that caused the failure. Table 3.1 enumerates the possible repair suggestions and indicates what types of failure each suggestion can help repair.

3.3.1 Solution alteration suggestions

Solution alteration suggestions propose repairing a failed goal by altering a solution to another goal. These suggestions amount to moving a light out of the way to make room for a second goal. The suggestion can be to move a light's hang point or focus point (potentially causing a change in hang point). If such a move renders a light unnecessary as part of the goal's solution the suggestion can also propose the removal

Table 3.1: Possible suggestions to repair a failed goal. The suggestions are listed from right to left in order of severity, i.e. how much they change the goal being modified. I_B and I_C are the border intensity and crossover intensity preferences, respectively, na is not applicable.

	Modify Solution		Relax Goal					Remove Goal
	Move	Move & Remove	I_B	I_C	Light Type	Direction	Area	
Not Enough Lights	na	x	x	x	x	na	x	x
No Hang Position	na	x	x	x	na	na	x	x
Direction Bound	x	x	na	na	na	x	na	x

of that light. Solution alteration suggestions change what hang points are available in the theater.

Solution alteration suggestions that only move a light can be used to repair a situation in which the direction bound for a goal could be met because the acceptable hang positions were occupied. Moving the light occupying the acceptable hang position frees up that position for use by the failed goal. These suggestions can also free up lights for use by other goals when slight changes of position and focus points allow for the removal of a light. It is important to note that solution alteration suggestions are made only when the proposed alteration does not cause a goal to fail.

3.3.2 Goal relaxation suggestions

Goal relaxation repair suggestions change either a goal's desired outcome or a goal's preferences.

When the direction bound for a light was not met, there is only one option–change a direction. Changing the failed goal's direction allows the light to be hung somewhere else. Changing the preference of a light occupying an acceptable hanging position moves the occupying light, freeing up a useful hanging position.

When there are not enough lights to satisfy a goal or no hanging positions exist, the number of lights currently being used must be reduced. This reduction can take

place when some goal in the system changes its light type, area, or its mechanical preferences (i.e, the border and crossover intensity).

3.3.3 Remove goal suggestions

Remove goal suggestions propose the removal of a goal and its solution from the list of goals that iPlot is trying to satisfy. This suggestion may be needed in the case when there are simply too many goals for all of them to be simultaneously solvable. Rather than reducing every goal and getting a less-than-desirable solution to every goal, a designer may choose the eliminate an entire goal, thus freeing up a large number of lights and hanging positions for use by other goals.

3.3.4 Representing repair suggestions

Each repair suggestion consists of three elements: the failed goal, a goal that will be affected if the suggestion is carried out, and the suggestion of what to change in the effected goal or solution. In the case that the suggested change is a relaxation of a goal, then the suggestion is an iPlot generated goal that represents the relaxation of the goal being modified.

Repair suggestion objects also provide the user with information about the repair's effect and the reason that it was proposed to the user. Knowledge of a repair's effect allow iPlot to determine how many light will become available if the suggestion is appied—useful information when iPlot is choosing repairs to carry out. An important feature of iPlot is its ability to provide a description of the repair in terms that a lighting designer can understand.

Chapter 4

Control Structure

Given the representations previously described for a performance space, lights, goals, failed goals, and repairs, iPlot must manipulate these objects making use of the knowledge they represent to form a solution–a group of lights that satisfies each goal. To accomplish this task iPlot uses a generate, test and repair cycle. Thus iPlot will propose solutions, test them for validity, and, if necessary, refine the goals and repeat the process.

4.1 Generating a proposed solution

Given a set of lighting goals iPlot has three separate kinds of generators that together can create proposed solutions for each lighting goal: light plot generators, light group generators, and light position generators. When each of these generators is created they are given a specific performance space in which they will be generating solutions. The relationship between these generators is shown in Figure 4-1. We present a brief overview of these generators before describing them in detail.

At the highest level is a light plot generator which takes a goal list and handles generating solutions for each goal, starting the repair process if necessary. The *light plot generator* takes as input a list of goals, and outputs a performance space object in which all lights forming the solution have been hung[1]. To find a solution for

[1]Thus whenever a light is "hung" we mean that it has been placed, i.e. hung on a pipe, in the

each goal the light plot generator first assumes independence between goals[2] (but not solutions). Under this assumption the plot generator in turn makes use of a light group generator that generates a set of lights that satisfy a single goal.

A *light group generator* takes as input a single goal, and outputs a group of lights that satisfy that goal. It also adds these lights to the current performance space object, which represents the evolving solution. The generator has a variety of different generation techniques that can be used to create a proposed solution. It chooses the technique to use based on the goal preferences. Each technique is a different method finding positions for a set of lights overlapping according to the preferences described in Appendix C.2. The group generator is concerned only with overlapping lights and uses a light position generator to find the best hanging position for individual lights.

A *light position generator* takes as input a light that needs to be placed, a direction, and a way of describing where the focus point of the light should be. The light position generator returns the best hang point for the light in the performance space. It does not hang the light. As a side effect, the light position generator changes the focus point of the light it was passed to be the focus point corresponding to the returned hang point. This will allow the light to reason correctly about illumination levels on stage after it has been hung.

Each of these three generators uses different domain knowledge to generate a proposed solution. Light position generators use knowledge about the relationship between a light's hang and focus points, and the resulting angle of the light beam to find the best match between light position and a direction. Light group generators focus on the knowledge about combining multiple lights. They combine lights using the goal preferences such as crossover intensity and border intensity, laying them out as specified by a generation technique. The plot generator is concerned with the validity of each proposed solution, the interaction between goal solutions and the repair process.

---

performance space as part of a solution

[2]The assumption for goal independence was justified previously in Section 2.1.

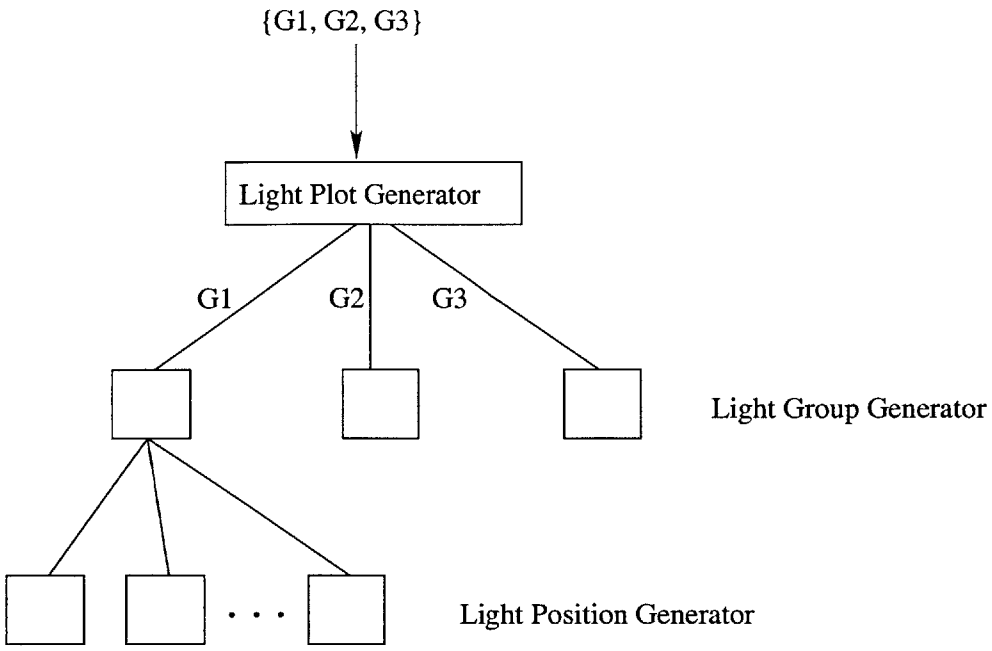


Figure 4-1: Solution generation overview. A single *light plot generator* takes as input all of the lighting goals to be satisfied. It in turns uses a single *light group generator* to generate a solution to each goal separately. The group generator in turn uses a *light position generator* to find available hanging positions in the performance space.

4.1.1 Light Plot Generator

The light plot generator is concerned with generating a solution to every lighting goal. Currently, iPlot assumes that all lighting goals are independent and generates solutions for each of them individually.

The first stage of this cycle is to generate a solution to each goal. The light plot generator uses a light group generator to generate a proposed solution to each goal. The solutions are generated according to their goal order, such that goals that the user specifies first are satisfied first.

This iterative solution generation takes place within a testing framework. After each proposed solution is generated it is tested against the hard constraints described in Section 3.2.2. If a goal fails, the plot generator ensures that the goal object's state reflects this failure and it's cause. The failure of a single goal does not stop the generation process, rather the failed goal is saved for later consideration.

After the plot generator has attempted to find a solution for each goal it looks at the collection of goals. If there are no failed goals, the plot generator terminates having succeeded in generating a complete set of lights that satisfies all goals and is hung in the returned performance space object. Otherwise, iPlot proposes repair suggestions for each failed goal. The user may then manually choose which repair suggestions should be followed or allow an automated repair strategy to direct the repair. If the user is directing the repair process they can manually start the next iteration of the cycle, otherwise the plot generator will automatically start the next round of generation when the automated repair finishes. These repair facilities are explained fully in the next section on repairing a failed goal.

4.1.2 Light Group Generator

The light group generator attempts to find a group of lights that satisfies a single goal. To do this, it must find a group of lights that together illuminate the entire area specified by the goal. The light beam from each light in the solution must come from the specified direction relative to the focus point of the light. This light group generation is guided using many of the preferences specified as part of the goal.

Each light group generator makes use of a light position generator, specified when the group generator is created. Using this light position generator, the group generator finds hanging positions for individual lights used as part of the proposed solution. The most basic generation methods of the group generator use the light position generator to generate a single line of lights whose combined light beams span the goal area. More complex generation methods are built upon this line generation method to create several lines of lights that together illuminate the entire goal area. Thus the process used to generate any group of lights to satisfy a goal will be a recursive process of using different generation methods, that at the most basic level place a single line of lights. In summary, the overall process of generating a proposed solution to a goal is as follows:

1. Choosing the type of light to use

2. Break the goal area into smaller sub-areas

3. Recursively generate solutions for each sub area. Recursion stops at the level of a single line of lights (potentially a single light).

All of the methods implemented in iPlot's light group generator follow these three steps. Currently iPlot implements total of six generation methods: two ways of generating a single, vertical line of lines that spans the area and four methods for generating a set of lights that illuminate an entire area. Figure 4-2 shows the group generators different methods and the dependencies between them.

We will first go through how iPlot chooses a generation method, light type, and then explain the generation methods in detail. Readers unconcerned with the details of this recursive generation process may skim through these sections or skip ahead entirely to Section 4.1.2.

Choosing a generation method

While iPlot's light group generator has many generation methods, the plot generator calling upon the light group generator need only ask for a solution to a given goal, they do not have to explicitly specify the generation method. The light group generator uniquely determines the generation technique using the goal generation preferences described. If the generation start point is a corner, the group generator uses the area generation from corner method. Otherwise, if a center point is given the generation technique preference is used to determine which of the three center generation methods should be used.

Choosing a light type

The first step in finding a group of lights that satisfy a goal is choosing the type of light to use. The light beams of the different categories of lights have very specific qualities, and the features of the lights themselves vary. Designers usually know what category of light they want to use to satisfy a single goal. But within each light type there are a variety of different lights made by several manufacturer and with a

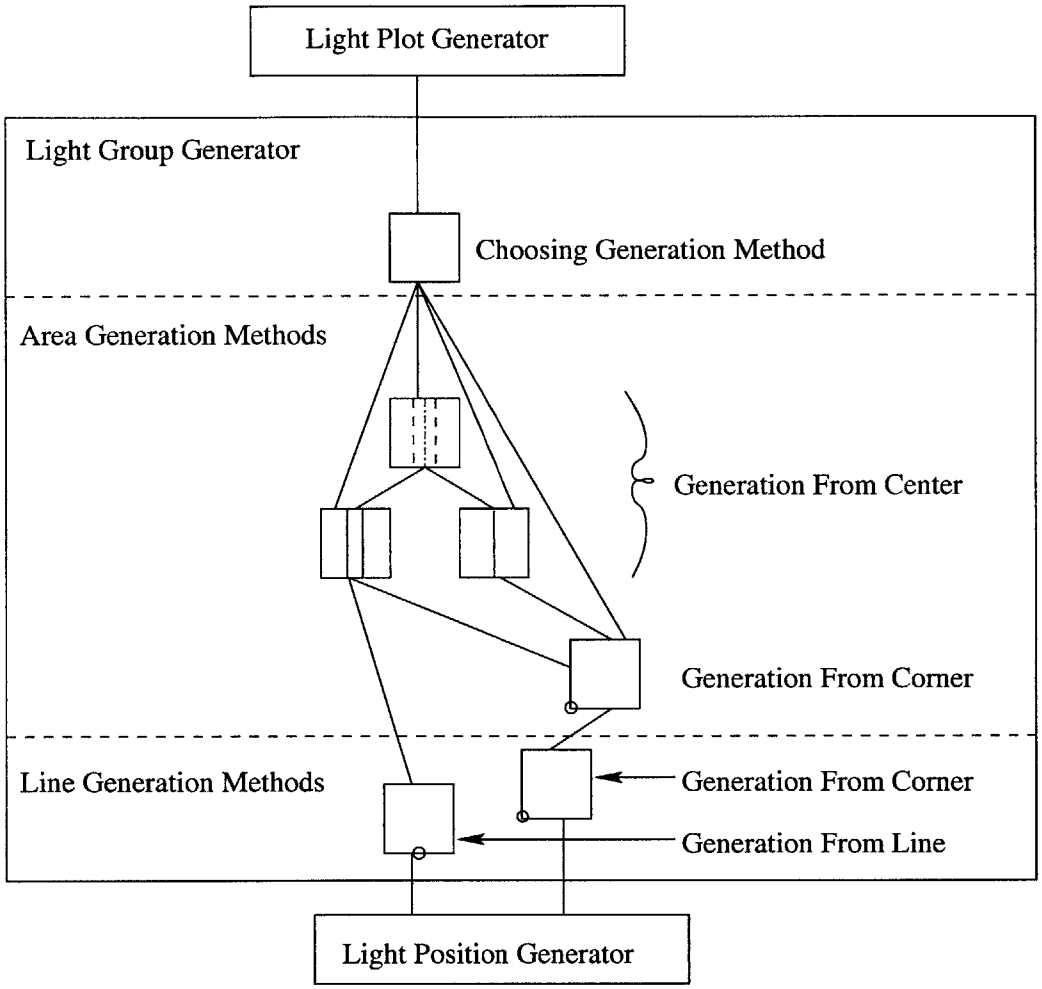


Figure 4-2: Group generator method overview. This figure shows that the plot generator calls a single method to get a group of lights which form a solution to a goal. This method in turns calls the appropriate area generation method. Each method makes calls to simpler generation methods to cover sub-areas, eventually reducing down to methods that generate a line of lights using light position generators.

wide array of beam spreads. To accommodate this situation iPlot allows designers to specify light type by specifying either a type of light to use, or a category of light and discretization[3].

When a designer specifies a category of light such as ellipsoidal reflector spotlight, the system must determine which light type best meets the requirements of the goal. This task is accomplished by testing each possible type. A focus point is chosen inside the goal area and for each light the best hanging position is found. The discretization obtained by each light type is then estimated from this single light. The light type with the discretization closest to that specified by the designer is used.

To estimate the discretization iPlot uses two distances from the test light. The distance from the focus point of the light to the crossover intensity point (crossover distance), and to the border intensity point (border distance). We assume that the lights on the border have their border intensity point exactly on the border. Given this assumption we can calculate n, the number of lights needed to span that stage, as

$$n = 1 + \frac{W-2B}{2C}.$$

Where B is the border distance, C is the crossover distance, and W is the width of the stage. This heuristic can be derived graphically from Figure 4-3.

area generation from center methods

The group generator's most complex generation methods are the methods that generate a group of lights that illuminate an entire area and whose focus points are symmetric around the stage. We choose this as our example of complex generation methods because it is one commonly used by lighting designers. A front light full stage wash, i.e. a set of lights that illuminate the whole stage and coming from the house (where the audience is), is most often generated in this way. The group generator of iPlot has three separate methods for generating a group of lights that are symmetric about the center of an area.

[3]Described in detail in Appendix C.1.

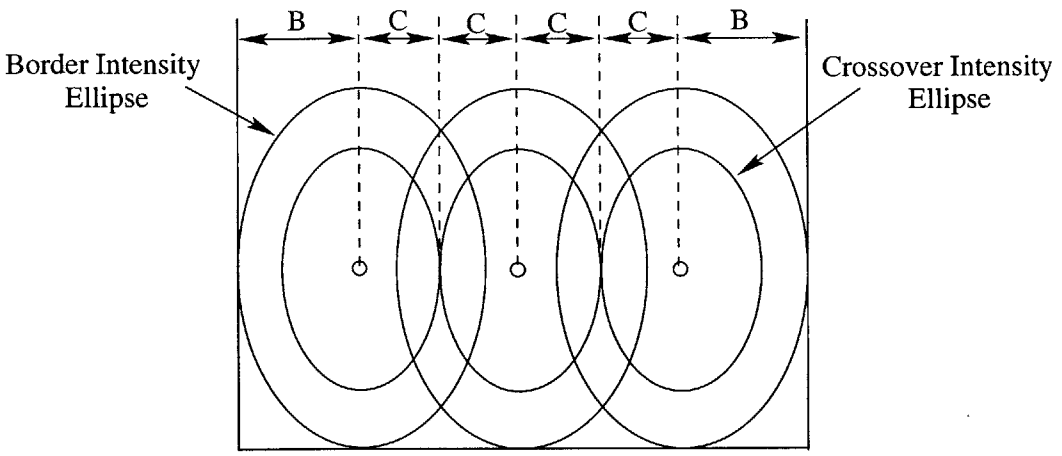


Figure 4-3: Estimating the number of lights to span the stage. As we can see it takes three lights to cover the width of the area. Because the crossover intensity circles of each light just touch each other, we can say that each light contributes $2C$ towards covering the width of the stage with the exception of the two lights at the borders, which cover $B + C$. As such $width = n \cdot 2C - 2C + 2B = 2C(n-1) + 2B$, which when solved for n produces our estimate $n = 1 + \frac{W-2B}{2C}$

Two of these methods are different ways of generating a group of lights symmetric around center. The first method generates a group of lights that is symmetric around the centerline of an area but does not have any lights whose focus point is on the centerline. The second method generates a symmetric group of lights that does have a line of lights whose focus points are on the centerline. Because of the symmetric nature of these algorithms the first method will always use an even number of lights to span the stage and the second method will always use an odd number. The third center generation method is an intelligent way of choosing whichever of the first two methods uses the fewest lights. Thus a designer who doesn't care if there is a light exactly on center can use whichever method conserves lights. Using the same process outlined in Section 4.1.2 for estimating the discretization of a light, we can estimate the minimum number of lights to cover the width of the stage. If this number is odd, then we want to use the second center generation method. If it is even, we want to use the first center generation technique.

The first method, symmetric about center not including center, breaks the goal area into two equally sized sub-areas, a stage left area and a stage right area. The

stage left and stage right sub areas are generated using a simpler generation method, generating an area from a corner, which is explained in the next section. Figure 4-4 shows this arrangement of sub-areas. By generating the two symmetric sub-areas using the same method, working in opposite directions the focus point on either side of the stage should be symmetric, assuming the hanging positions are also symmetric about center[4].

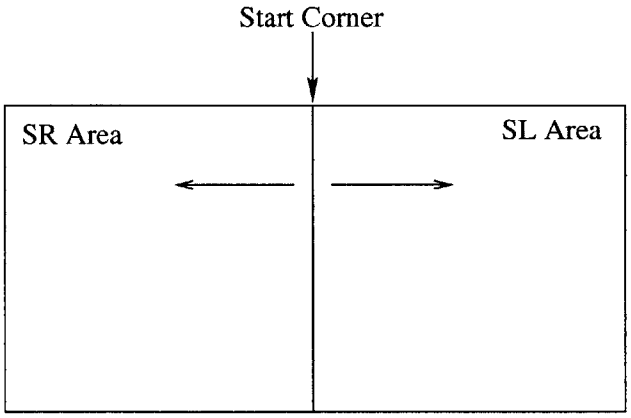


Figure 4-4: Area generation from center, no lights on center. The stage is divided into two symmetric sub-areas, each of which is generated using a generate area from corner method, starting at the upstage or downstage center point.

The second of these methods generates a group of lights whose focus points are symmetric about the center of an area including a line of lights on center. This is done by first generating a vertical line of lights centered on the centerline of the stage. The unlit portion of the stage is then divided into two equally sized, symmetric about center areas, which are covered using the corner generation methods using techniques similar to that of the first generation method. These areas are shown in Figure 4-5.

area generation from corner method

The area generation from corner method is group generator's simplest generation method that searches for a set of lights that can illuminate an entire area of arbitrary

[4]Most theaters have regularly spaced pipes at least as wide as the stage, centered on center stage such that this assumption holds. Even if the pipes are not laid out symmetrically, this method will generate an area that is symmetric as possible.

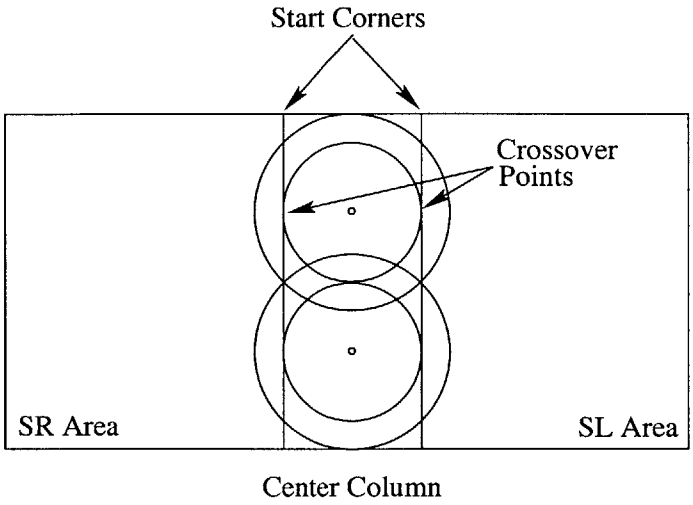


Figure 4-5: Area generation from center, lights on center. The stage is divided into three sub areas: a central column and two symmetric sub areas, one on each side of the column. The column is a single line of lights generated using a line generation from line technique, the other sub areas are generated using from an on-stage corner

size. The corner generation method generates divides the stage into columns, each of which can be illuminated by a single line of light. The lights to illuminate each column are generated using the line generation from corner method described in the next section.

For the purposes of explaining the corner area generation method let us assume the designer has just given a goal for a front light full stage wash, symmetric about center without lights on center. The generator has just started generating the stage left sub-area using vertical lines of lights.

The corner method starts by generating an initial column of lights, starting at the corner specified when calling this method. In this case the corner is down-stage center. After this first column has been laid down the generator tests to see if the border opposite the start point, the stage left border in this case, is sufficiently illuminated. The generator considers a border sufficiently illuminated if the intensity at the border is at least as bright as the border intensity preference.

If the border is not sufficiently illuminated, a second column of lights is generated adjacent to the first. To determine the amount of overlap between columns generator

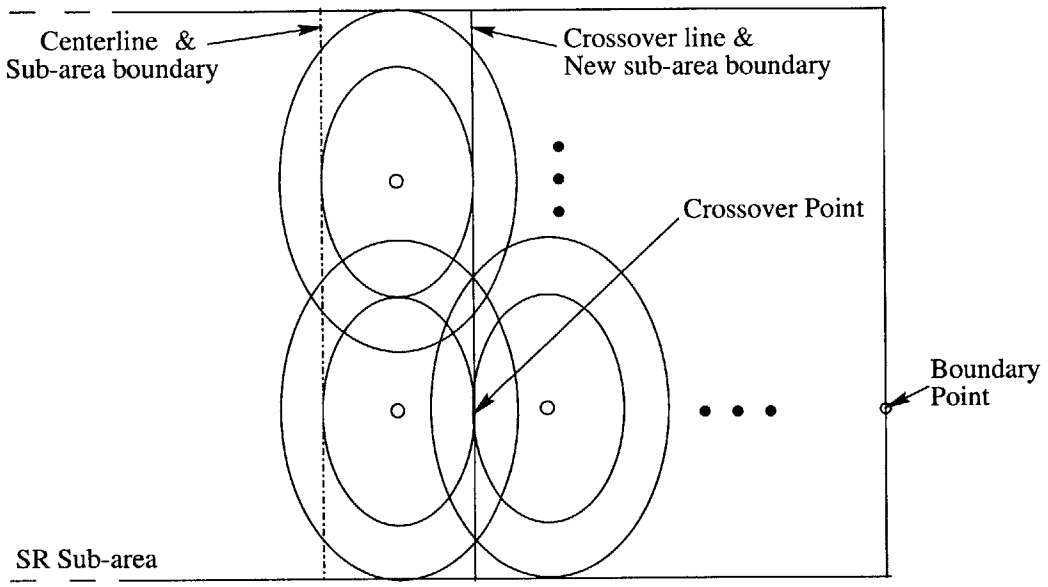


Figure 4-6: Area generation from corner. The generation of the stage left sub-area is shown above. Starting center stage the generator places columns of lights until a column sufficiently illuminates the boundary point.

looks at the first light in the newly placed column and finds its stage left crossover point[5]. Because we want crossover points to overlap, this crossover point is on the new boundary for the next column of lights. After laying down the second column the border is again tested. This process of test and then generate columns repeats until the border is sufficiently illuminated and the process stops.

line generation methods

The line generation methods are used internally by iPlot as a building block for more advanced generation methods. They generates a set of lights whose focus points all lie on a single line. The combined light pool from these lights span the stage. As we have seen both the center and corner generation methods reduce their problems to generating a line of lights. We have implemented the methods to generate vertical lines of lights running upstage and downstage. This process can be generalized easily to generate lines of lights in other directions.

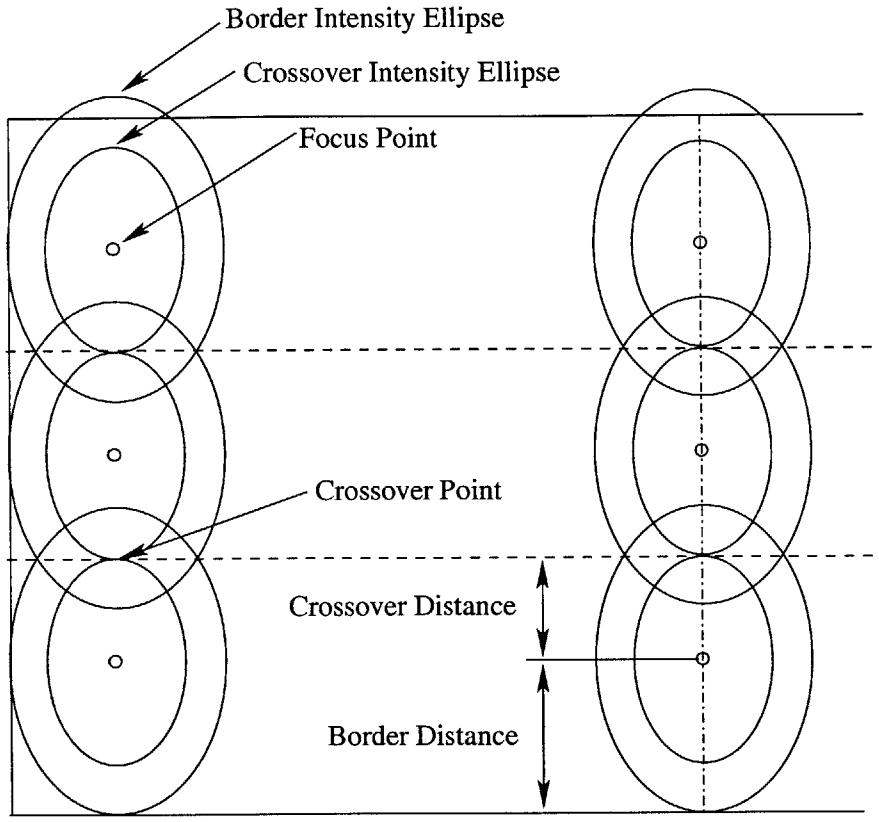
[5]As mentioned in Section 3.1.3 given the crossover intensity and a direction, in this case stage left, the light returns the crossover point. The details of how this is done are covered in Appendix B

Generating a vertical line of lights is done using a goal area and either a specified bounding edge (generating from a line), or bounding corner of that area (generating from a corner). It is important to note that while the line generation methods take as input an area, the combined light pool from the line of lights will span the goal area but there is no guarantee that the entire area will be illuminated.

We have seen examples of situations where both of these line generation methods are used. When generating a wash that was symmetric around center with lights on center, the line of center lights in generated from a bounding edge, the downstage edge of the stage with the lights centered on the centerline. The wash to either side of this central column was placed using the corner generation method which exclusively uses the line generation from corner method.

Let us again consider generating the stage left sub-area of our symmetric full stage wash. Our first column of lights starts just stage left of downstage center and goes all the way to the upstage edge of the stage. The first step is to remove a light from the inventory for use, and find the best hanging position for it. We want the centerline to be bordering the crossover ellipse of the light beam and the downstage edge of the stage to be bordering the border ellipse of the light beam. We can pass this information, along with the first light and direction to a light position generator and get the best hanging position for the light and hang it in the performance space. The light position generator conveniently sets the focus point of the light to where it needs to be given our intensity bounds.

After this first light has been placed we then test if the upstage edge of the goal area is illuminated sufficiently by this light. If it is, we stop, otherwise we generate the next light in the column. In this case we find the upstage crossover point of this first light. The y-coordinate of this crossover point is the new y-coordinate of the light position generator downstage boundary line for the next light. We can then pass the light position generator a second light, the direction, and the new boundaries and get the optimal hang position. This place and test process repeats until the border is sufficiently illuminated, Figure 4-7 shows this placement process graphically.



(a) Corner Generation Technique (b) Line Generation Technique

Figure 4-7: Line generation. Lines of lights are generated by successively overlapping lights along a line so that their crossover points are in the same location until the boundary is sufficiently illuminated. Part (a) shows how light placement can be bounded by two lines forming a corner, while part (b) shows how light placement can be bounded using a boundary line and a centerline.

Hanging the lights of a light group

In describing these light group generation methods we said that the line generation methods take lights from the inventory and hang them in the performance space as positions are found. This was a slight simplification. Lights are hung in the performance space by calling a method in the performance space object to hang a light, passing the hang point and the light to this method. As each light is hung in the performance space as part of a goal solution two additional actions occur. First, the light is added to a goal's list of lights forming the solution, second, the light is told that it is satisfying that goal.

Failure during light group generation

A light group generator can fail to completely satisfy a goal when a light position generator fails or their are not enough lights of the appropriate type to illuminate the entire stage. In either case generation stops and the reason for failure is propagated up to the light plot generator. This reason will be a violation of one of the hard constraints mentioned in Section 3.2.2 along with any addition knowledge such as the number of lights needed to complete the goal, or the closest available hang position that was out of bounds. In the case of a failure all lights that were hung prior to the failure remain in the performance space and associated with the goal.

4.1.3 Light Position Generator

The light position generator is concerned with finding the best hanging position for a single light. Corresponding to the types of areas on a pipe, iPlot can search for ideal, forcible, or available hanging positions. This process can be broken down into three steps: finding the focus point; finding the best hanging position on each pipe given a focus point; and then comparing the hanging positions available on each pipe.

Finding the focus point

When lighting designers place a light they do so in one of two ways, either they know exactly where they want the light to be, i.e. they directly know the focus point, or they know how bright they want the light to be at some point and roughly what direction the center of the light should be relative to that point. If the focus point is known, then the position generator just finds the best hanging position given that focus point and a direction. When the focus point is not known, the generator must search for the correct focus and hang point pair such that the known point is illuminated at the specified intensity.

In the case where the exact focus point is not known then the designer, or the position generator, must search for the proper focus point. This is done by estimating a first focus point and finding the corresponding hang position for the light, testing to see if it meets the intensity requirements, and if not moving the focus point and repeating the process. The position generator has two different methods for searching for a focus point: line bounded search and corner bounded search. Figures 4-8 and 4-9 summarize this search process, the details can be found in Appendix D

Finding the best hang position on a pipe

Given a focus point, a direction, and a pipe the position generator can find the best position on a pipe to hang the light. The "best" position to hang a light is the position that creates the focus line that is closest to the direction when the direction is centered on the focus point. Under these conditions the point on the pipe that has the smallest relative angle compared with the direction is the best hanging point. Figure 4-10 shows a hanging position and it's relative angle with respect to a direction.

Directions are capable of reasoning about the point with the smallest relative angle out of a list of points, and pipes can enumerate the ideal, forcible, and available hanging positions on the pipe. Thus, the position generator finds the best hanging position by connecting these two components. It obtains potential hang points of the appropriate type from the pipe and hands them to the direction asking for the hang

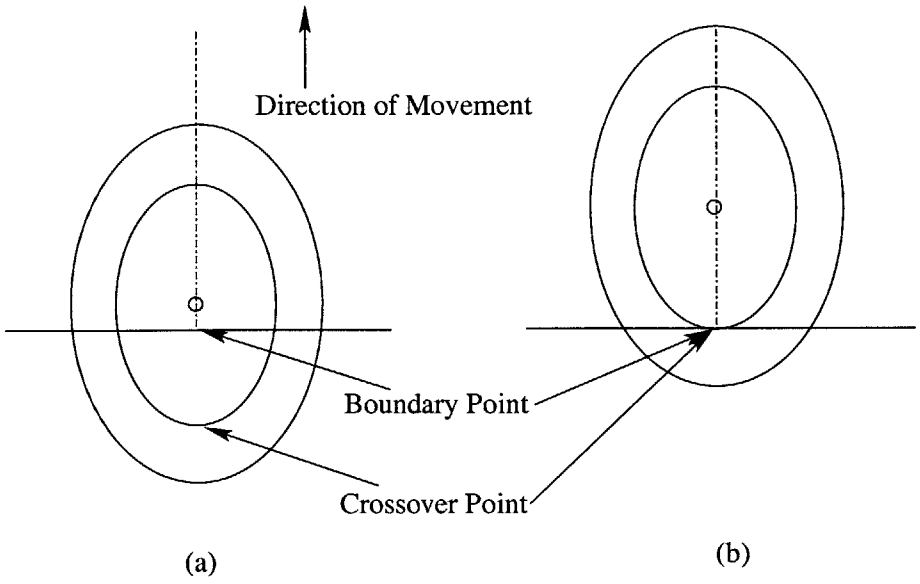


Figure 4-8: Finding a light's focus point from a line. A light beam's position is specified a bounding line and a point on that line to be at a specified intensity. The focus point is then found by choosing focus points, testing the intensity at the bounding point, and if necessary moving the focus point such that the intensity at the bounding point approaches the desired intensity.

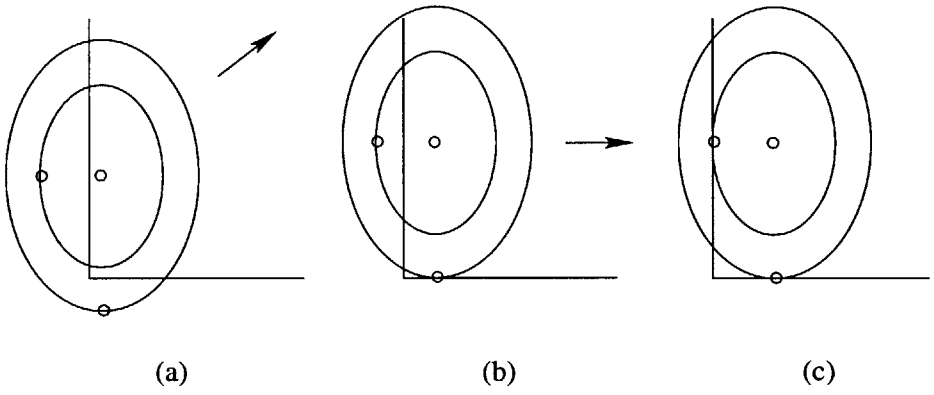


Figure 4-9: Finding a light's focus point from a corner. A light beam's position is specified by two bounding lines, each of which must be at a specified intensities. The focus point is then found by choosing focus points, testing the intensity at both points, and if necessary moving the focus point such that the intensity at the bounding points approaches the desired intensity.

point with the minimum relative angle. The returned hang point is the best one for that point.

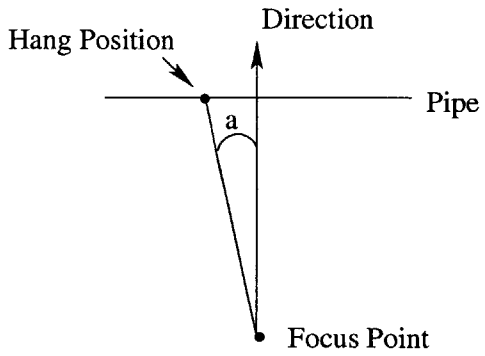


Figure 4-10: Relative angle between point and direction. We can see that the relative angle between the hang position and the direction is a. By inspection one case see that in this case the hanging position with the smallest relative angle is going to be at the intersection of the pipe and the direction.

There is one subtlety to this process and that is the definition of the "minimum" relative angle. In comparing relative angles between a point and a direction, the designer may choose to distinguish between the xy-angle and the z-angle. If the goal specifies that the ideal, non-prioritized angle is what a designer cares about then the minimum relative angle is just that, the absolute minimum relative angle, attempting to minimize both angles simultaneously. If, however, the designer chooses to prioritize an angle, let us suppose the xy-angle, then the point with the "minimum" relative angle is that with the minimum relative xy-angle. If two or more points have this minimum relative xy-angle, then their z-angles are compared. Directions are capable of reasoning about points with both definitions of minimum, and the choice of which type of reasoning it use is determined by the goal preferences.

Comparing pipe positions

After obtaining the best hanging position on each pipe for a light, the position generator can find the best overall position through a similar process using the direction. Now instead of a collection of points on a single pipe the generator uses a single point from each point and picks the one with the minimum relative angle to the direction

centered on the focus point.

Returning the best position

After the light position generator has determined the best hang position for a light it returns that light position to the method which called upon the generator. In addition it sets the focus point of the light passed to the generator to the focus point that corresponds to that hang point. This ensures that after being hung the light has the correct hang and focus points from which to base it's photometric calculations.

Failing to find a position

A light position generator can fail for two reasons: either the best hanging position does not meet the hard constraints of the angle bounds, or no hang position exists at all. When this happens the light generator fails and propagates the reason for failure to the method caller.

4.2 Repairing a failed goal

We say that a goal has failed when a proposed solution meeting all of the hard constraints could not be found. This situation occurs for one of three reasons: there were not enough lights available to complete the goal, the desired goal direction could not be achieved, or there were no hanging positions available in the theater[6]. After a goal has failed the first step in the repair process is to create a list of repair suggestions, each of which proposes to "repair" a goal by doing one of the following:

1. Modifying a proposed solution
2. Relaxing a goal
3. Removing a goal

[6]In most situations a lighting designer usually runs out of lights to hang before completely exhausting the set of all hanging positions. As such, this failure does not happen in practice.

Note that choices two and three result in a new goal set. Using an explanation of why a particular goal failed, iPlot's repair suggestion generator proposes a set of repair suggestions, each of which will, when carried out, result in the failed goal no longer failing for the same reason[7] (i.e., violating the same hard constraint). The choice of which suggestions to carry out can be made by either the designer or by iPlot using an automated repair strategy. After carrying out the selected suggestions, iPlot generates a new solution to this alternate set of goals.

4.2.1 Solution dependence and goal relaxation

The generation of a new goal set is unusual in such a computer-aided design tool. Most such systems generate new solutions. The reasoning behind iPlot's approach is as follows. When some goals in a goal set cannot be satisfied we can approach the problem of fixing this situation in one of two ways. We could attempt to find the set of solutions that is closest to satisfying the goals, but may not satisfy them. Alternatively, we could ask: What is the set of goals closest to the original goals that I could satisfy. In theory approaching the problem from either direction should result in the same solution. The difficulty lies in defining what makes one proposed solution "closer" to satisfying a goal than another proposed solution, and how you define nearness between goals themselves.

In iPlot, and generally in lighting design, the solutions are very dependent on each other, regardless of goal interdependence. This situation derives from the fact that all the solutions occupy the same physical space, the theater, and are competing for resources in that space, i.e. lights and hanging positions. Because of this, solutions that iPlot creates are very dependent on goal ordering. The fewer goals that are satisfied in the performance space when a new solution is being proposed, the more resources available to be used in that solution. This competition for resources means that in many cases it is easier to approach the problem of finding a good solution by modifying the set of goals. iPlot implements a hybrid strategy that considers

[7]There is no guarantee that a relaxed goal will not fail when the solution for it is generated, only that it will not fail for the same reason.

modifying both goals and solutions.

4.2.2 Proposing Repair Suggestions

Repair suggestions represent a way to increase the chances of a failed goal being satisfied. It is the job of the *repair suggestion generator* to propose a set of repair suggestions for each goal. For each failed goal the the generator will propose repair suggestions by:

1. Looking up the type of suggestions that will repair that goal, determined by the goal's failure explanation.
2. Determine what goals should be modified by repair suggestions.
3. For each possible goal and suggestion type pair, propose the appropriate repair suggestions.

Determining type of suggestions to propose

As originally presented in 3.3, iPlot's knowledge about which types of failure are repaired by whichsuggestion types can be stored in a table, presented again below as Table 4.1.

Table 4.1: Possible suggestions to repair a failed goal. The suggestions are listed from right to left in order of severity, i.e. how much they change the goal being modified. I_B and I_C are the border intensity and crossover intensity preferences, respectively, na is not applicable.

	Modify Solution		Relax Goal					Remove Goal
	Move	Move & Remove	I_B	I_C	Light Type	Direction	Area	
Not Enough Lights	na	x	x	x	x	na	x	x
No Hang Position	na	x	x	x	na	na	x	x
Direction Bound	x	x	na	na	na	x	na	x

Thus, given a failed goal, the system uses the information in Table 4.1 to find the repair suggestion types that can be used to repair this particular goal. For each type of suggestion the system then finds all goals or goal solutions that can be modified with a suggestion of that type to fix the failed goal.

Determining what goals to modify

What goals can be modified to repair a failed goal depends on the circumstances of the failure. For each of the three failure reasons one of two conditions existed in the performance space: either the goal is impossible to satisfy in the performance space, or it would normally be satisfiable, but a previous solution is using the necessary resources (lights or hanging positions). In the first case, the only way to repair the failed goal is to modify the failed goal itself. In the second case, the failed goal could be modified, or whatever goal (or solution) is using the necessary resources could be modified. When it is possible that modifying another goal or solution in the system could repair a failed goal, the repair suggestion generator determines which goals as follows.

When the failure is due to not having eough lights, we want to make more lights of the failed goal's light type available. Thus, all suggestions will in some way cause a goal to use fewer lights. Suggestions can be generated for every goal in the system that uses the same light type as the failed goal.

When the failure is due to not having any hang positions, we want to free up hanging positions. This freeing up is also accomplished by making some goal use fewer lights, but in this case there is no restriction on which goal, and potentially every goal in the system could be modified. The repair suggestion generator, however, reasons about what goals would be best to modify. The repair suggestion generator looks at where the failed goal's solution would be hung if possible and only suggests modifying the solutions (or goals) that result in those desired hang positions being occupied by other lights.

When the failure is due to a direction bound not being met, then there is a light occupying every acceptable hanging position. Again the system can look at where the

failed goal's solution would be hung if possible and suggests modifying all solutions or goals that result in those desired hang positions being occupied by other lights. It is important to distinguish that while the goals being modified are similar in this case to having no available hang positions, any suggested goal relaxations will be very different (changing direction rather than intensity or area).

Proposing the repair suggestions

At this point iPlot has a set of goals that can be modified, and a set of repair suggestion types and must propose a repair suggestion of each type for each goal. The last step then is to determine what the exact change will be. For goal removal suggestions, this is trivial; the only possible change is removing the goal. For the other suggestion types, the state of the performance space must be analyzed to predict the effect of each change.

It is important to note that the current implementation of iPlot does not know how to predict the effect of all possible repair suggestions. Of the possible repair suggestion types iPlot can predict the effect of moving lights within an existing solution, changing the border intensity of a goal, change light type, and changing the direction of a goal. We limit our discussion to how these four types of changes are predicted.

Move Light Suggestions

Move light suggestions are made when moving light can make useful hanging positions available for the lights of a failed goal. This situation occurs when the goal failed because a direction bound was not met. We must find the hang positions needed by a failed goal to determine which, if any, of the lights occupying those positions can be moved without causing previously satisfied goals to become unsatisfied. Recall that lighting pipes are broken down into three areas: available, forcible, and unavailable. Forcible was defined as am area on the pipe currently occupied, but the occupying light can move without causing any goal to become unsatisfied. Distinguishing between forcible and unavailable areas can be done by moving the occupying lights along

the pipe and testing if they still satisfy their goal's direction bounds[8]. Furthermore, light position generators can be asked to find ideal (ignoring all lights currently hung in the performance space) or forcible hang position.

Thus a move light suggestion is generated in two steps. First the light position generator is used to find the best forcible hang position. If the failed goal light can be hung at this point and satisfy its goal, a suggestion is then made to move the light currently occupying that position along the pipe and hang the failed goal light. Otherwise, no suggestion is generated because moving any light would cause a previously satisfied goal to become unsatisfied.

Change Border Intensity Suggestions

A change in border intensity has the potential to reduce the number of lights needed to satisfy a goal. This reduction can occur when a subset of lights forming a solution overlap the border, but their intensity is not sufficient to meet the border intensity requirement. Thus, additional lights were needed to increase the amount of illumination at the borders of the area. In this case, reducing the border intensity will make these additional lights in the solution unnecessary. A situation such as this is shown in Figure 4-11.

While there are technically an infinite number of border intensities, minute changes in intensity have little effect. Designers rarely consider changes of less than five percent, and so we use this value as the smallest change in intensity to consider. As a result we can consider a finite subset of changes. For each of these potential changes in border intensity we can estimate the effect of changing the border intensity. This is done in two steps. First count the number of lights whose light beams' border ellipse intersect the the area boundary given the current border intensity. Then, count the number of lights whose beams' border ellipse intersect the area boundary will the

[8]If a moved light is still within its goal's direction bounds after the move, iPlot assumes that the light beam has not changed significantly, and the goal is still satisfied. If the moved light's effective direction would no longer be in the bounds of the goal direction, then it cannot be moved. In this way the direction bounds are limiting the number of times we can "nudge" a light assuming it won't effect the solution. It is bounding the transitivity of light movement.

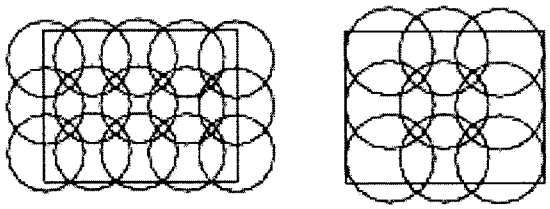


Figure 4-11: Estimating the effects of changing goal area border intensity. The left image shows the border intensity ellipse of the light beams projected onto the goal area after a solution has been generated. While the area is currently covered using a set of 15 lights, we can see the area is only slightly larger than the area covered by border ellipse of the inner square of nine lights. The right image shows the predicted change if we modified the goal to cover the same area, but with a lower border intensity. The lights would not change position, but the size of their border intensity ellipses increases. As a result only nine lights are needed to cover the area.

new, lower border intensity. The difference between these two counts are the number of lights that would not be needed if we changed the border intensity.

For each goal for which we want to generate a border intensity suggestion, we use the above method to find the largest border intensity that will require fewer lights than the current solution. A suggestion is made to change the goal's border intensity to this estimated intensity. It is important to note that no change is made to make the border intensity (less than or) equal to zero. With a border intensity of zero all borders would always be sufficiently illuminated. As a result, after the first light was placed and the area borders examined, generation would stop because all borders (and thus the entire area) are sufficiently illuminated.

Change Light Type Suggestions

A change light type suggestion is exactly that—a suggestion to change the light type of a goal. It can be used when a goal failed because there were not enough lights of a specific type. In this case the light type of all goals with the failed light type can be changed to any other light type, so long as there are enough lights of that type available. Thus for each goal with the failed goal light type iPlot looks at all the

light types in the inventory. If the beam spread of the light is wider than the beam spread of the current light type and there are at least as many in the inventory as use in creating the solution, then a change light type suggestion is made. If the beam spread is smaller than the current beam spread iPlot can estimate the number of lights needed to cover the stage in a manner similar to that of estimating the discretization, it can estimate the number of lights needed to span the area in both directions. The product of those two light counts is the number needed to cover the entire area. If that many lights are available in the inventory of the type under consideration, a change light type suggestion is made for that goal.

It is interesting to note that we do not consider this repair suggestion valid for goals that fail because no hang position available. It is the only suggestion that is used to repair failures due to not enough lights and not failures due to no hang position available. This situation arrises because while changing the light type will presumably free up hanging positions for a failed goal, the goal whose light type is changing will most likely then fail—either because again no hang positions exist, or the one or two hang positions that are available would not let the needed lights satisfy their direction bounds.

Change Direction Suggestions

Change direction suggestions are made whenever all the hanging positions that are acceptable for a light in the failed goal are occupied. This situation prevents the failed goal light from being hung in a position that allows it to meet its angle bounds. In this case though we are looking at relaxing the goal by changing the desired outcome, in effect changing the ideal hang position of the lights. To find the desired change in direction iPlot first uses the position generator the best available hang point for the failed light goal. Then iPlot finds the relative angle between the available hang point and the direction. At this point a suggestion can be made to increase the bounds of the failed goal's direction by this relative angle, making the available hang point acceptable. The suggestion is not changing the desired direction, but rather just

saying that the available hanging position should be considered acceptable.

Now for each goal that could be altered to repair this goal (i.e. including those occupying acceptable hang positions) iPlot also proposes a suggestion to move the entire goal direction by the relative angle. This change in desired direction will result in different occupied hanging positions, allowing the failed goal light to be hung at an acceptable position.

4.2.3 Repair Strategy

In general it takes only a few of the proposed repair suggestions to allow all goals to be satisfied. The blind application of every suggestion would result in sub-optimal solutions. As an example, consider the case where there are two goals G1 and G2 that use the same light type. G1 is satisfied and G2 failed because there were not enough lights. There will exist a suggestion to change the light type of both goals, and to remove goal from the system. Applying any one of these suggestions would ensure that the next round of generation could find a solution to all goals. Applying all of them would remove all goals from the system, leaving the worst solution possible–none at all.

It is the job of the repair strategy to choose the subset of repair suggestions that will be carried out. To generate a single "optimal" repair strategy would be difficult because every designer is going to have different preferences towards how they would like to repair a set of failed goals. Much of the information they might use in user-directed repair might not be available in iPlot. For example if a goal could not be satisfied the designer's first repair choice might be to replace it with a completely different goal. Currently iPlot has no facility for expressing this situation. As a result, the most basic repair strategy is to generate an a set of repair suggestions and present them to the designer. The designer can then choose the subset of suggestions he wants followed, in addition to making any changes or additions to the goal set that aren't suggested.

While every repair situation is going to be slightly different, we believe that differ-

ent designers have distinct styles or strategies for repair that iPlot can model. Thus iPlot has an interface for creating automated repair strategies that given a list of goals, failed goals, and repair suggestions will return the subset of suggestions that should be followed. Currently iPlot has implemented one such automated repair strategy based on the preferences of a single lighter designer, the author, called simply the Perelson repair strategy.

We shall provide an overview of the Perelson repair strategy on a set of four goals: G1, G2, G3, and G4. The Perelson repair strategy assumes that the goals have carefully been prioritized by the designer and these priorities are reflected in the goal order. Thus G1 is the most important goal.[9]. The Perelson strategy allows no goal to modify a higher priority goal as part of a repair. Thus all repair suggestions that modify G1 to fix G2, G3, or G4 are discarded. Furthermore, the strategy allows only one repair suggestion to affect each goal. Thus if a suggestion to repair G2 by modifying G1 is chosen, no other suggestion that repairs or modifies G1 or G2 is followed[10]. Because of this strict limitation on the number of suggestions that will be followed, the strategy picks repair suggestions for the higher priority goals first. Thus it will first look at suggestions that repair G1 then G2 and so on.

With these broad policies in hand the Perelson strategy chooses specific suggestions to be followed. First any solution modification suggestions are chosen. Then, in the case that a goal failed to meet it's direction and the repair suggestions are to change a direction, the suggestion to change a goal's own direction is followed. Thus if G2 failed because it's direction bounds could not be met, the only suggestion followed would be to change G2's direction. When the cause of the error was not enough lights the Perelson strategy looks to change the border intensity; change the light type; or a remove a goal, in that order. Thus if G3 failed because it did not have enough lights the strategy would first attempt to reduce it's border intensity to as

[9]For those interested the author would have chosen G1 to be a front light wash of the entire stage. This ensures that at the very least the audience will be able to see everything on stage clearly.

[10]This is done because multiple modifications of one goal would interact. Trying to free up lights by changing both the crossover intensity and light type of a goal is counter productive. All lights will be released by changing the light type and changing the crossover intensity would be unnecessary. iPlot does not attempt to predict which repairs can safely interact.

low as 30%. If the border intensity is already this low, or such a change is predicted to have no effect then the system looks to change the light type of G3 or G4 (if it uses the same type of lights and contains lights in its solution). As a last resort if previous methods have not resulted in a useable suggestion then a remove goal suggestion will be chosen, essentially saying the set of goals is over-constrained and the only way to fix the problem is to remove a goal, thereby relaxing a large number of constraints at once.

4.2.4 Goal refinement and regeneration

After generating repair suggestions for all failed goals and pruning those suggestions using a repair strategy, iPlot must set up for another cycle of generate, test, and repair. The first step in this process is to make a copy of the current, unrepaired set of goals and solutions. This step allows the user to back-up and undo a set of repairs if he does not like the next iteration. After this step all lights are removed from the performance space object, clearing all the old solutions from the physical model. Now the system is ready for a new set of goals, so iPlot carries out each of the repairs selected by the repair strategy. After carrying out all of the repair suggestions we are back to the original state of the system: There is a set of goals that needs to be satisfied. The plot generator takes this set of goals and starts iteratively generating them, and the cycle continues until either all goals are satisfied, or the user chooses to stop the process and accept an intermediary state as an acceptable solution.

Chapter 5

System Evaluation

To evaluate iPlot we had it find solutions for a set of realistic goals, and presented these goals and corresponding solutions to a designer for analysis. The designer was asked to comment on the solutions as if iPlot were one of his design students and he was giving us feedback. After analyzing these prepared solutions the designer was asked to try using iPlot for himself and to give us verbal feedback as he experimented with the system.

We present here three goal sets and corresponding solutions that demonstrate some of iPlot's generation and repair abilities. We present a case with a goal whose solution found by iPlot required repair and was considered a good solution by the designer; a case in which there is only one goal that was considered satisfied by iPlot, and whose solution was judged less than satisfactory by a designer; a case in which there were two conflicting goals that were repaired and the solution was considered good by the designer.

5.1 Evaluation of three solutions

5.1.1 Example 1: Single goal, a good solution demonstrating self repair

iPlot was given the goal shown in Table 5.1

Table 5.1: Goal for Example 1

		Value	Bounds
Direction	xy-angle	DS	±5°
	z-angle	45°	±20°
Crossover Intensity		0.6	±.05
Border Intensity		0.4	±.05
Discretization		$\frac{1}{5}$	na
Light type		ERS	na
Start Point		USC	na
Technique		Fewest Lights	na
Automated Repair		Yes	Perelson Strategy

iPlot generated a solution using two iterations of the generate, test, and repair cycle. The generation method chosen by iPlot was to generate an area symmetric around center, with lights on center[1]. We can see from Figure 5-1 that iPlot successfully generated the center column of lights and was in the process of generating the stage left subarea when it failed. By inspection (and from iPlot's output) we can see that it failed because no available hanging position allowed it to illuminate the remaining portion of the stage left subarea from the specified direction. At this point iPlot generated three goal relaxation repair suggestions: change the direction bounds to allow an available hang point to be acceptable, change the direction itself so that all lights can achieve that direction, and remove the goal. The Perelson strategy dictated that the direction bound change be followed as the least severe change to the goal. After this repair suggestion was carried out, iPlot tried to generate the solution to this relaxed goal, this time succeeding. The solution is shown in Figure 5-2

The designer upon seeing this solution commented that overall the solution was very good. The repair chosen was the one he would have picked had be been directing generation, and in fact the one he would have used had he been satisfying the goal without iPlot. Overall he believed that the illumination on stage would actually look more even than shown because such a wash of lights would normally get diffusion added to it, which scatters the light a little making it fade out more gradually.

[1]See Section 4.1.2 for the details of this generation technique

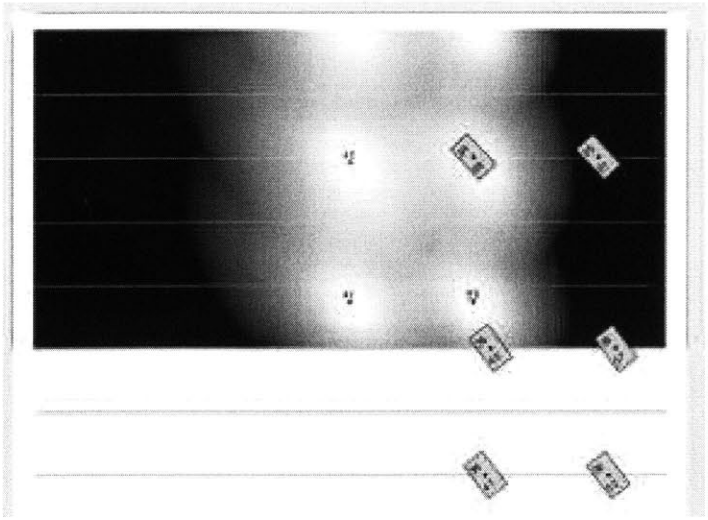


Figure 5-1: Example 1: Good solution part 1. At this point the generator has failed because there is no hang position available that will allow the direction bounds to be met for a light illuminating the rest of the stage left area.

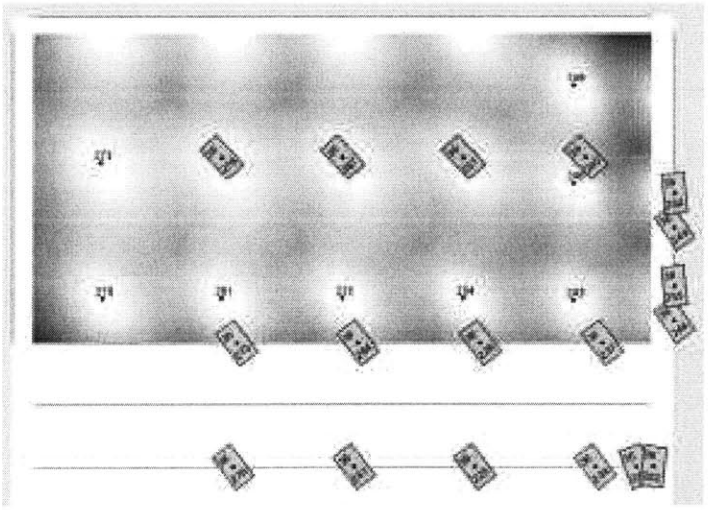


Figure 5-2: Example 1: Good solution part 2. After using the Perelson strategy to choose a direction change suggestion to repair the failed goal, iPlot comes up with this solution to the new relaxed goal.

5.1.2 Example 2: Single goal, a bad solution

iPlot was given the goal shown in Table 5.1

Table 5.2: Goal for Example 2.

		Value	Bounds
Direction	xy-angle	DSL (40° SL of DS)	±°
	z-angle	45°	±20°
Crossover Intensity		0.6	±.05
Border Intensity		0.6	±.05
Discretization		$\frac{1}{5}$	na
Light type		ERS	na
Start Point		DSC	na
Technique		Fewest Lights	na
Automated Repair		Yes	Perelson Strategy

iPlot generated this solution in one attempt without needing any repair. It chose to use the generate area from center, no lights on center method as the one that would require the fewest lights. This solution is shown in Figure 5-3. The designer, however, upon seeing this solution did not think it was very good. He pointed out that while the entire goal area was lit, both the upstage and downstage borders could be illuminated better. To demonstrate his point he showed what the solution would look like with only one row of the lights simulated as being on, as shown in Figure 5-4. He pointed out that the downstage row of lights is focused primarily below the stage where the light is not needed. Furthermore, a designer would typically use a shutter on the light to prevent the light from illuminating any area offstage. Such a large shutter cut would not be possible in this case and, even if it were, doing so would dim the light considerably as the shutter would be blocking the majority of the light beam. The designer continued by saying that a better solution would be to move all of the lights upstage by a pipe, moving their focus points upstage by the same distance. This change would cause more of the downstage row of lights to be on stage. As a result both upstage and downstage borders would be brighter, and a smaller shutter cut would be required on the downstage edge of the area.

iPlot generated this bad solution because of its use of the generate area from corner technique, starting upstage center. iPlot placed the top row of lights such that

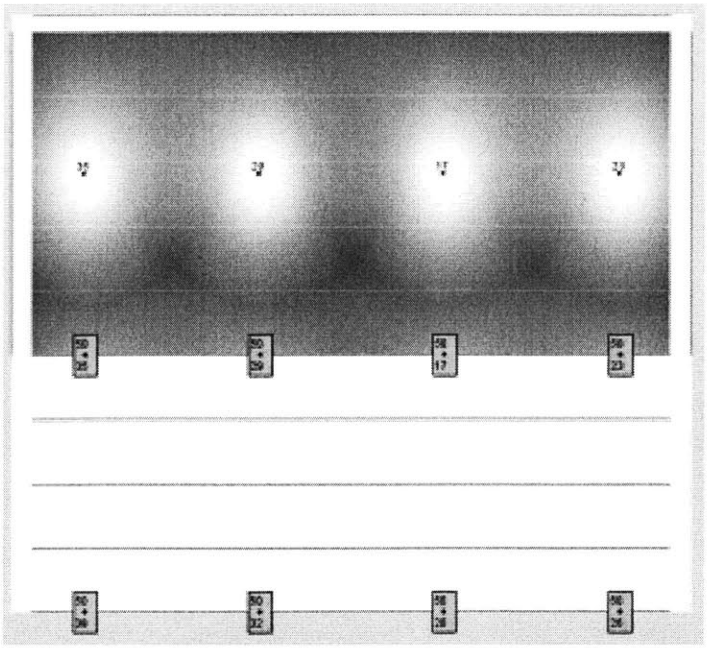


Figure 5-3: Example 2: Bad solution part 1. iPlot successfully generated this solution to two goal 2, however, it was not considered acceptable by the designer.

the border intensity ellipse of each light just overlapped the upstage border of the stage. With this top row position set, the crossover preference dictated how close the downstage row was, which in this case was far below stage. After generating this solution iPlot checked only that the solution it generated did not violate any hard constraints, that it covered the entire goal area from the appropriate direction.

This problem in generation could be fixed in one of two ways. We could increase our knowledge of generation techniques to incorporate a better method of laying out lights, or we could test for non-optimal strategies and attempt to repair them. A smarter generator could make an estimate of how many lights would be needed to span the stage both horizontally and vertically, and use this to predict the best placement of the first set of lights relative to the border. With our goal for Example 2, iPlot could have reasoned that it would only take slightly more than one light to cover the entire area, and that therefore there is no need to start with the top row of lights as far from the border as possible. Instead it could estimate a closer acceptable focus point, which would result in an optimal placement of lights. If instead we

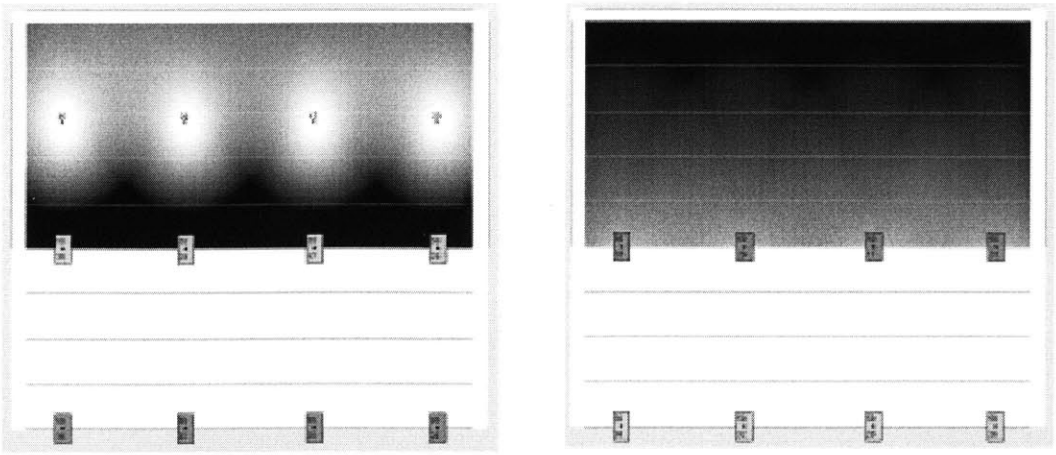


Figure 5-4: Example 2: Bad solution part 2. On the left we see the solution with only the top row of lights on, and on the right we see only the bottom row of lights on. As we can see from the image on the right, the solution would be improved if we moved all of the lights to the next pipe upstage, moving their focus points the same distance.

turned to a repair mechanism to fix this problem then iPlot would need knowledge of how to recognize non-optimal solutions. As we've seen from this example, lights whose focus points are far outside the goal area are a good indication of non-ideal solutions because a majority of the light beam is not being used. After recognizing this situation iPlot could then propose a repair that shifted the lights such that the lights that focused off stage now focused on stage.

5.1.3 Example 3: Multiple conflicting goals, a good solution

iPlot was given the goals shown in Table 5.3. The inventory contained: 15 Source Four 26° lights, 15 Source Four 36° lights, and 50 Source Four 50° lights.

Table 5.3: Goals for Example 3.

		Goal 1		Goal 2	
		Value	Bounds	Value	Bounds
Direction	xy-angle	DS	±5°	US	±5°
	z-angle	45°	±20°	60°	±20°
Crossover Intensity		0.6	±.05	0.6	±.05
Border Intensity		0.4	±.05	0.4	±.05
Discretization		$\frac{1}{5}$	na	$\frac{1}{5}$	na
Light type		ERS	na	ERS	na
Start Point		USC	na	USC	na
Technique		Fewest Lights	na	Fewest Lights	na
Automated Repair		Yes	Perelson Strategy	Yes	Perelson Strategy

iPlot generated this solution using two iterations of the generate, test, and repair cycle. The generation method chosen by iPlot for both goals was to generate an area symmetric around center, with lights on center. iPlot also chose both goals to use Source Four 36° lights. We can see from Figure 5-5 that iPlot successfully generated the first group of lights. It had finished generating the center column of lights for the second goal and was in the process of generating the stage left subarea when it failed. By inspection (and from iPlot's output) we can see that it failed because there were not enough Source Four 36° lights available. This is because goal 1 is using the majority of the 36° lights[2]. At this point iPlot generated four goal relaxation repair suggestions. For each goal it suggested changing the light type to Source Four 50° (which had enough lights to cover the entire area), and removing the goal. It is interesting to note that no border intensity change suggestions were generated. All smaller border intensities were estimated to require the same number of lights. The Perelson strategy dictated that light type of the second goal be changed because it had a lower priority and changing light type is the less severe of the two suggestion types. After this repair suggestion was carried out, iPlot tried to generate the solution to this relaxed goal, this time succeeding. The solution is shown in Figure 5-6.

[2]Note that here we are seeing an example of how solutions are very dependent because of resource contention and goal ordering comes into play in determining what goals are initially satisfied.

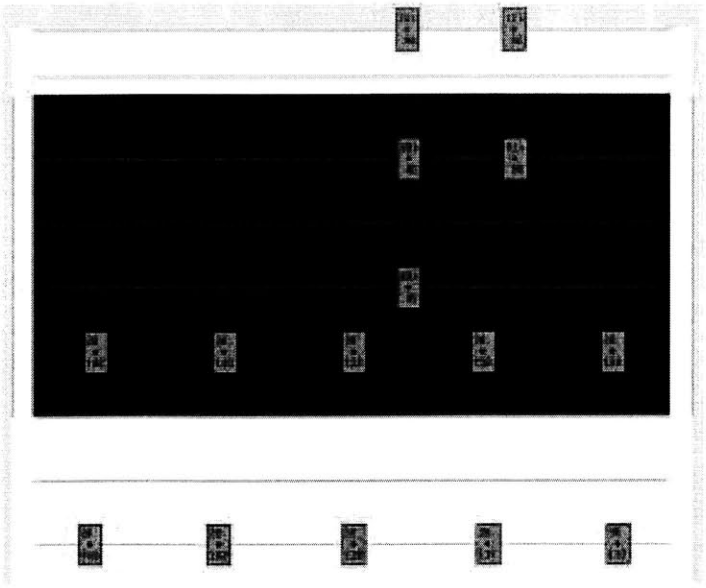


Figure 5-5: Example 3: Multiple conflicting goals solution part 1. At this point the generator has failed having used all of the available Source Four 36° lights before it had successfully satisfied goal two. Because all Source Four 36° lights are in use, one of the goals must change their light type to be satisfied

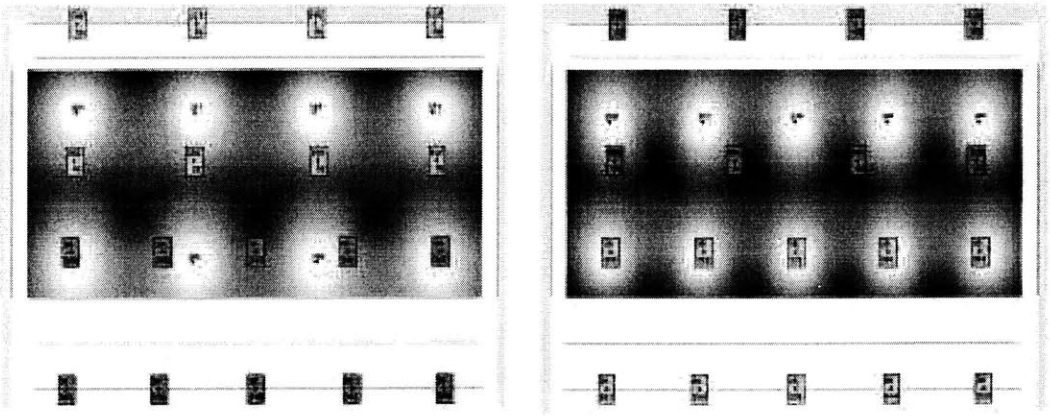


Figure 5-6: Example 3: Multiple conflicting goals solution part 2. As the lower priority goal, the back light light type was changed to use Source Four 50° and both goals were successfully generated.

Similar to our first example, the designer upon seeing this solution commented that overall the solution was very good. The repair chosen was the one he would have picked had be been directing generation, and in fact the one he would have used had he been satisfying the goal without iPlot.

5.2 General Feedback

During his experiments with iPlot, the designer gave us some feedback on the system as a whole. Overall he liked the effect that the bounds in direction had, but thinks they should be made even more complex. He said that while designers have a range of directions they may think are acceptable, they want all lights within a solution to have the same direction. Thus the designer's direction ideas might be modeled more accurately as a bounded range model such as the following: A direction as a whole can be given a bounds, for example $\pm 15°$, but within that bounds they would like all the directions to consistent to within some smaller range, such as $5°$. The designer also pointed out that while this is rule generally true, like all rules it has exceptions. For example, the situation presented in Example 1 above is quite common. Here the designer does not worry as much about the lights at the boundaries of the area being within the range of the bounded range model because they realize it would be impossible.

The designer also mentioned that the situation we saw with iPlot in Example 2 was not uncommon. Fairly often iPlot would generate a solution that while accomplishing it's goal was not ideal. Currently iPlot stops searching when it finds an acceptable solution, not checking to see if there might be a more acceptable solution. We believe that iPlot could become better at handling this situation in three ways. First, the generator itself could become smarter, perhaps by adapting the start point by small amounts. Second, using the same machinery that should be developed to make move and repair lights suggestions, iPlot could analyze existing solutions to see if the can made better. Third, iPlot could make quick methods of adapting solutions available to the designers themselves.

The designer also mentioned that he believes that certain designers may want to increase the number of preferences controlling the overlap. He believed there are situations in which the designer would want independent control over vertical and horizontal crossover intensity, as well as the border intensity for each border. He cited as an example the case in which much of the action of the play takes place down stage, near the audience. In this case the designer will most likely want the downstage border bright, but may not care if the other borders are as bright.

Overall the designer thought iPlot needed a bit more work to become extremely useful, but that it was a very good start. The designer said that he could imagine using iPlot quickly as a brain storming tool to see what might be possible with a given inventory, even if he did not directly keep iPlot's solutions. While iPlot's solutions may not be perfect, they allowed the designer to find an acceptable solution quickly. This, in turn, let him think about how the solution could be changed slightly to improve it, a faster process than starting from scratch by hand or using computer drafting tools.

Chapter 6

Related Work

At this time we know of no other intelligent design tools that focus specifically on the domain of lighting designer. This lack of availability was in fact one of the motivating reasons for iPlot. There are, however, many commercially available tools designed to aid lighting designers and researches have been looking into other aspects of lighting design. We will present a brief overview of the other work that has been undertaken in the lighting design domain. Then we will present some information on related intelligent design tools from domains such as architecture.

6.1 Lighting Design

6.1.1 Commercially Availably Lighting Design Tools

There is quite an extensive base of software aimed at aiding certain steps in the lighting design process. Computer aided design and drafting programs such as AutoCAD [1] and VectorWorks [8] are widely used to draw and render two- and three-dimensional representations of both lighting and scenic designs. They do not, however, support, lighting design per se. They represent a physical model with no notion of design goals or preferences. Software such as Beamwright [4] or Rosco's Lighting Paperwork System (LPS) [13] include a large selection of lights' photometric properties and can calculate area covered, intensity, etc. for an arbitrary angle and distance

from a light. In addition there are many programs to help lighting designers organize the paperwork that represents their lighting design, e.g., Lightwright [5], LPS, or VectorWorks. There is an increasing trend towards encapsulating all of this functionality within one program. LPS and VectorWorks have had much new functionality added with each new version. In addition, lighting control consoles themselves have become more advanced, now including rendering and paperwork tracking capabilities. Electronic Theater Control's (ETC) Emphasis Control System is such a system [3]. The software listed here represents only a few of the most popular products within each category.

6.1.2 Lighting Research

Research in computer graphics, specifically three-dimension rendering, has focused on realistic renderings of theatrical lighting [2]. Such systems give lighting designers the ability to evaluate their designers prior to their implementation in a performance space. These systems take as input the location of light sources, light source properties, and other spatial information. Thus this work is focusing on the step after iPlot, or a lighting designer, has completed the step of assigning lights to hanging positions and a more complete analysis of the light plot is desired.

System's such as Lula [12] address the programming of the lighting control console where designers consider individual light looks (a.k.a. cues) for each moment of a performance. This considers lighting design after the lighting plot has been designed, accepted, and implemented and now the designer must consider how to use each light during the performance itself. At this step designers are deciding what lights to use, how bright they should be, and how and when to transition between looks. This information is saved in a lighting console and later used again during the actual performance itself. Lula approaches the problem from a functional programming perspective and lets designers associate the implementation, the individual lights, with their purpose[1].

[1]In this instance purpose refers to what the lighting is representing or helping emphasize. For example, the a yellow light might be representing the sun itself, or it could be used as part of a set

6.2 Intelligent Design Tools

Other intelligent tools use search strategies similar to iPlot's, such as dependency-directed redesign [7, 6] or generate-test-debug [11].

iPlot's generate-test-repair strategy is closely related to the technique of dependency-directed redesign, which in turn combines the ideas of dependency-directed backtracking and generate-and-test (TAC)[7]. The technique was employed in a computer-aided architectural design tool called The Architect's Collaborator (TAC). TAC analyzes architectural designs, reasoning about how abstract qualities, e.g., visually open, are translated into a physical form, represented as a 2D floorplan[2] TAC starts with a set of design goals and a design provided by the user, evaluates the design with respect to the goals, and generates new designs. For designs that are not solutions, it proposes repair suggestions, and continues with its generate-test-repair strategy. TAC's search through the space of possible designs is dependency-directed, guided by dependencies between abstract qualities and physical form and the dependencies for a specific goals, which serve as explanations for unsatisfied goals. This search process is very similar to iPlot's. Both systems use the explanations for failed goals to narrow the search space. While TAC only uses this explanation to guide in altering solutions, iPlot uses this knowledge to alter solutions and relax goals.

Generate-test-debug is a paradigm used by Gordius [11]. Gordius is a planning system in that it attempts to find a set of events that can explain the formation of a geological region. While iPlot is not a planning system because it does not generate a sequence of steps to transition between states, it does share many similarities with Gordius. Both systems use the generate-and-test paradigm by generating a solution, testing it, and repairing if necessary. While iPlot uses the results of the tests to form both alternate solutions and relaxed goals, Gordius uses the results to fix parameter values within its plan steps. Also, Gordius prunes conflicting plan steps, much as iPlot's repair strategy chooses which repair suggestion to carry out.

---

of lights that are being used to create a scary atmosphere.

[2]While not directly relating to iPlot it is interesting to note that this translation is similar to an earlier part of the light design process when a designer is considering what lighting goals will best represent their abstract design concept.

Chapter 7

Conclusion

7.1 Future Work

This initial version of iPlot focused on creating a complete system that could map physical lighting goals to a set of lights, hang positions, and focus points for a subset of common theatrical spaces. Future work on iPlot should focus on a few key areas that would increase the usefulness, usability, and efficiency of iPlot. This work includes improving iPlot's existing models for the performance space, allowing a larger set of theaters to be modeled. Additionally, the domain knowledge incorporated into iPlot represents a small fraction of the knowledge available, incorporating more knowledge will allow for the creation of even better solutions. In addition, the current version of iPlot does not consider an important aspect of the light designer's process–their reliance on past experience. iPlot would benefit greatly from a case-based reasoning system that has the potential to enhance solution generation and repair.

7.1.1 Improving existing mechanisms

Applicability to all theater spaces and efficiency were not priorities in designing and implementing the three-dimensional space that serves as the basis of iPlot's physical model. To increase the situations that iPlot can be used in we must start relaxing the assumptions made about possible shapes. Currently iPlot only considers rectangles

with a constant z-coordinate value. This assumption is fine for representing many black box theaters, but not for oddly shaped performance spaces or most theater in the round[1]. To represent realistic stages and goal areas iPlot should be able to handle arbitrary n-sided polygons as well as ellipses in three dimensions. Similarly, lighting pipes should be modified to allow curved pipe sections as many pipes in the theater curve to follow the form of the performance space or the stage.

The initial restriction on shapes was made to simplify the algorithms to find hanging positions and layout a series of overlapping lights that cover the stage. Any modification of the performance space representation motivates a need to change these algorithms. It is also important to consider the running time of these algorithms and improve their efficiency because in the creation of one solution the knowledge to find hanging positions can be used hundreds of times and increasing the speed of these algorithms can have a dramatic effect on the search time[2].

In addition to the physical structure of the performance space, iPlot does not model all light types, or all the specific instruments manufactured of each type. Future versions of iPlot should consider more light types, such as fluorescent or HMI lights. For each light type, iPlot should have larger set of specific instruments from a range of the popular manufacturers.

7.1.2 Incorporating more domain knowledge

The first version of iPlot considers only a small portion of the knowledge used by experienced designers. Most notably iPlot could improve its techniques for modifying solutions and could include knowledge about other issues that effect the physical layout of lights.

[1]Theater in the round is when the audience is on all sides of the stage. As a result the stage is very often shaped like an ellipse

[2]As an example, at one point during the implementation of iPlot the algorithm for a line to find the point with the smallest relative angle from a list of points was improved to use binary search rather than linear search. This one change decreased the running time of the program to find solutions to two identical goals, full stage washes, by 30%

Solution repair techniques

Currently iPlot has a limited set of solution repair techniques. If two lights want to be in the same place it can slightly move them along the pipe. This knowledge alleviates some of the need to repair and regenerate entire goals. This knowledge could be generalized to handle the shifting of an arbitrary number of lights. As an example three lights that wanted to be in the same place should, if acceptable, be centered around where they want to be. This would mean one light in the precise position and the other two centered around it. Knowledge of goal order could be included in this process such that the light with the highest priority could be the light of the three that gets the ideal position. Besides moving on a pipe, it is occasionally possible to move lights to nearby lighting pipes and the solution repair mechanisms could include this notion.

Other issues effecting light placement

There are quite a few issues that arise in the design of stage lighting that iPlot ignores. Inclusion of these aspects of the domain would improve the quality of iPlot's solutions and it's usefulness to designers. Currently iPlot does not take into account the set, the physical structure of the performance space itself, the availability of circuits, or lighting accessories.

Representing the set and the physical structure of the performance space would allow iPlot to reason about the objects that can obstruct a light beam. Potentially the hanging position that results in the best direction is non-ideal because there is an object between the goal area and the hanging position. Because iPlot does not currently model any such objects, it is not aware if that situation happens. Adding these representations also would require that iPlot be able to reason about where obstruction of a light beam is acceptable and how designers might deal with it. Quite often lighting designers don't mind a light that is partially obstructed, as long as the obstruction is not illuminated.

Either in tandem with or after a lighting designer has placed all of the lights to

be used for a performance, he considers the availability of electrical circuits to power all of these lights. It is possible in many performance spaces, particularly older ones, to hang more lights than you have the ability to turn on independently or at all. Thus a few situations can arise. There may be enough circuits, but their placement throughout the performance space is such that there is not enough cabling to run power between the lights and the circuits. There may be enough power available, but less one circuit per light. In this case two lights must be powered by the same circuit, always turning on at the same time, effectively combining in many ways. At best this reduces the flexibility of a single solution, at worst designers must connect lights that do not satisfy the same goal. Because the two goals might be independent and the lights used at different times during the performance, in such a situation the designer might choose instead to reduce the number of lights used by one of the goals. Often the work of figuring out how to power individual instruments is delegated by the designer to a show's master electrician[3]. None the less, iPlot could increase its knowledge to assist in this aspect of the designer process.

7.1.3 Extension: case-based reasoning

Lighting designers draw greatly on past experience when creating a light plot for a new design. This reliance on past experience can be modeled by iPlot through the use of a case-based reasoning system. This system would have a record of performance spaces, shows, goals, and solutions that could be retrieved and adapted for use in solving and repairing current sets of goals. We will first explain in detail the motivation for a case-based reasoning system and how it could integrate with iPlot to create better solutions. Then we will go over some of the design issues and requirements that must be considered in creating a case-based reasoning system for iPlot.

[3]The theatrical master electrician supervises the implementation of a designer. They over see the hanging, circuiting, and focus of lights in the performance space.

Motivation

Designers use the knowledge gained from previous lighting designs in a variety of ways. At the very least, this experience helps the designer search more effectively because they have a better understanding of what is possible with a given inventory, or in a space in which they have worked before. In some situations, a designer may use complete solutions from previous shows in future ones. Following the use of previous work by a designer, a case-based reasoning system in iPlot would have several functions.

In one sense the case base can act as an archive for the designer, remembering the collection of lighting goals that make up the design for a specific show. The designer could use iPlot to review or revise shows worked on previously. This use of complete shows would be useful when a show is brought back to a theater, a situation known as a revival. A more likely use of the archive would be to bring back previous solutions. A designer may remember a specific lighting implementation of a goal from iPlot and want to re-create that look exactly, down to the repairs the goal underwent. In this case the exact solution could be retrieved and put into the performance space.

At times a designer may not be interested in recreating a goal exactly, but may think of one goal as a variant of another. There are a lot of preferences associated with the goals in iPlot, and each designer is likely to develop a few preference combinations that capture his style of lighting design. By retrieving previous goals designers can retrieve a set of preferences that closely match their style. The retrieved goal can then be modified slightly to match what the designer currently wants to do. Thus we believe that it will be easier for a designer to remember a specific instance when a set of preferences worked well, than to remember the exact combination of preferences. This situation is especially true if iPlot is extended to allow for even more preferences, thus improving generation and reducing the need for repair.

iPlot also could make use of the case base to recreate past solutions without the explicit direction of the designer. Consider that a designer could ask for a goal that had been previously solved without knowing it had been solved previously by iPlot.

This could certainly happen in environments where more than one designer uses the same version of iPlot. The light group generator would interface with the case base as a generation method. If the case base had a solution that could be put into the performance space, then the light group generator would retrieve the solution and use it, otherwise it would generate a solution from the physical model. This retrieval could also be done at the level of partial solutions if the group generator considered cases as sub-areas.

One could stretch the case-based reasoning system even farther and give it an ability to reason about solutions, giving it a way to infer cases that were never actually solved. Consider the case when two different goals have identical solutions. Let us consider G1 and G2, both of which are full stage front light washes. G1 and G2 are identical, except for the z-angle of their direction. G1 has a z-angle of 45° and G2 has a z-angle of 50° . Given the spacing of pipes in a theater, and that only a few z-angles are actually possible it is quite likely that these two goals have the same solution. From this we could further assume that a goal G3, identical to G1 except for a z-angle which ranged between 45° and 50° would have an identical solution without having to solve for G3 explicitly. A case-based reasoning system capable of this type of reasoning would over time develop a model of what was achievable in a performance space and be able evaluate new goals using that model to determine if their solutions would be identical to previous ones.

So far we have mentioned ways in which the case base is being used to put a solution into the performance space, but this is only one of the uses of the case-based reasoning system. With experience a designer also becomes better at generating solutions to completely new goals. Explained computationally, the act of creating past designs, or cases, allows lighting designers to develop better heuristics to estimate what a solutions to a goal will look like using a specific generation method. This is true even in situations when the designer is not exactly recreating a remembered goal or solution.

As an example we can consider two situations in which iPlot could make use of the case based reasoning system as a source of heuristics. Light group generators

can use previous cases to estimate the discretization achieved with a given light for a goal. This heuristic would improve the light choice given a set of possible lights, as discussed in 4.1.2. Previous cases also can serve as heuristics for the repair suggestion generator and repair strategy. Let us suppose we have the goal G5, which has failed because there were not enough lights. The repair strategy must choose between the various suggestions, each of which make a different number of lights available for G5. If G5 had been previously solved, perhaps in a situation when it did not have such a low priority, we could use this prior solution to estimate how many lights G5 needed to create a complete solution. This knowledge would help choose the repair suggestion that would help repair G5 and release the fewest lights from other goals.

Another example of using heuristics during repair is estimating the change in solution that would occur with a change in goal. Suppose to repair G5 we wanted to change the border intensity of the goal G4. If the case base has a goal similar to G4, but with a lower border-intensity the solution in the case base can be used a quick and accurate estimate of what would happen if the system were to change G4's border intensity.

Design Issues

Starting from the uses for a case based reasoning system in iPlot, we can begin to formulate ideas about what should be stored in the case base, how it should be indexed, and how it should interface with the rest of the system.

There are four types of information the case base will need to store and cross reference: performance spaces, shows, goals, and solutions. Designers will work in a variety of performance spaces in which iPlot may be used to solve lighting goals. Each set of goals is used to create the design for a single show, thus a show is a reference to a number of goals that were solved together. Each goal was generated for a specific performance space and specific show and had a single solution. Each solution is associated with a single goal. It is important to consider each of these pieces of information and their relationship to each other. Without all four it is impossible to uniquely identify the situation in which a solution was used. Because of solution

dependence the same goal will have a variety of different solutions, potentially one for each show in which a goal appeared. Potentially it also might be possible that a solution and goal are applicable to two performance spaces because of a similarity in stage size and potential hanging positions, but the solution might not be optimal in both places. As a common example consider that performance spaces have moveable pipes, which can be configured differently for each show. Two different configurations of a performance space in iPlot would be considered two different spaces, but they would be remarkably similar. Many solutions could be implemented in both, but because of the slight changes in available positions not every goal will have the same ideal solution.

Having determined what must be stored in the case-based reasoning system it is also important to consider how each piece of information should be indexed and retrieved. Each object should get a unique identifier that can be used to retrieve it. This capability will allow for a compact representation of the relations between objects. For example, a goal in the case base could store unique ID triplets (performance space, show, solution) to identify each situation in which a solution to that goal had been found. Alternatively, one could have a goal store just a list of solution unique IDs, but each solution in the case base would know the performance space and show that it was associated with. Because such unique IDs span multiple uses of iPlot and would be used primarily by the case-based reasoner, these IDs could be created by the reasoner itself.

For many of these items the system also will need a way of mapping between unique IDs and a reference that the designer can remember, most likely as something east to remember, such as a string. Performance spaces and shows both typically have names, and each goal is for a specific purpose. There is no guarantee that these names are unique, however. Considering combinations such as show and performance space will reduce the overlap but it might be necessary to include a concept of time. It is doubtful that a theater is going to have two distinct shows of the same name in the same performance space at the same time. Indexing using this information could be performed entirely at the level of the user-interface or built into the case

base itself.

It is also important to realize that there are times when the system will not want to exactly reference a specific goal or solution. Consider the times when a case is to be used as a source of heuristics rather than for an exact solution. In this situation there is no specific show being referenced, just a performance space and a solution. The case-based reasoning system could have a variety of methods that could report on the number of matching solutions, retrieve a specific one, or all of them to the iPlot component interfacing with it.

There also will be times when both the designer or iPlot components will care only about certain features of a goal. The designer may, for example, want to see all goals that cover a specific area with a single direction, but compare the different preference choices. Similarly, a repair strategy might want to look at all solutions to a goal that vary only in border intensity when making suggestions. To accomplish this there could be a type of goal template that is used to describe what features are desired in retrieved cases, and what features can have any value. It could even consider ranges of features such as directions within a set range of directions.

Before implementing a case-based reasoning system in iPlot we must decide upon specific answers to the questions raised here about storing and indexing performance spaces, shows, goals and solutions in iPlot. This work will be undertaken in the next version of iPlot.

7.2 Summary

Because of iPlot's informed search process using generate and test, it is able to efficiently search the space of possible light arrangements for a set of lighting goals. It was found useful as a tool by lighting designers to quickly brainstorm about possible lighting arrangements, allowing them to consider many more options than they would have otherwise.

iPlot has demonstrated the effectiveness of its goal relaxation technique as part of the repair process. This technique sets iPlot apart from most other generate and

test systems, which consider only modifying solutions.

Because of these contributions iPlot is a step toward an intelligent tool for conceptual lighting design.

Appendix A

Geometric Constructs

Within the three dimensional space of the theater iPlot uses basic solid geometric shapes as the building blocks of its representation of objects. The basic elements it uses are points, lines, rectangles, directions, and dimensions.

A.1 Point

A point in iPlot represents one point in the discrete coordinate space with a specific x, y, and z coordinate. These points can be translated in each dimension, effectively changing the point that is being represented. Furthermore, any point can calculate the distance between it, and any other point in the coordinate space.

A.2 Line

A line, or more accurately a line segment, in iPlot is defined by its starting point and ending point, and represents all points that lie on the line between those points. A line is capable of several basic operations: it can calculate its length; tell if it contains a specific point and even enumerate all of the discrete points that lie on the line. A line can also reason about the relative angle between it and a specific point; more specifically it can reason about the angle between two lines: it and the line defined by its starting point and the specified point. This reasoning is extended into reasoning

just about the relative angle only on the XY plane, or XY-Z plane. Building upon this knowledge, a line given a list of points can choose the point that forms the smallest relative angle with that line.

A.3 Rectangle

A rectangle in iPlot is a simplified three-dimensional rectangle defined by its upstage right corner, width, and depth. Rectangles in iPlot are flat, with all points having the same z-coordinate, in effect being a two-dimensional rectangle that can vary in height. This is a very limiting assumption that was done to simply the geometric algorithms needed to reason about the rectangle. Within iPlot, rectangles are used solely to build representations of the stage, which more often than not is a two-dimensional rectangle. Further versions of the system should work on relaxing this assumption. Rectangles are only capable of reasoning about their size and what points are contained within the rectangle.

A.4 Direction

A direction in iPlot is a direction vector which represents two angles: a xy-angle which represents the angle off the positive x-axis formed by the vector, and a z-angle which represents the vertical angle off the xy-plane. Directions can be centered on a point, such that all reasoning done is about directions whose origin is that point. In essence a direction is a specialized kind of line in which the starting point is the point on which the direction is centered, and the ending point is an approximated end point generated from the two angles.

Because of this representation, direction objects are able to reason about relative angles between a point and the direction. This is useful in reasoning about lighting placement. A direction is one of the two components of a lighting goal. Given a direction and a focus point for a light, one can center the direction on the focus point and evaluate various hanging points by comparing their relative angles with the

desired direction. The point with the minimum relative angle is the hanging point the best achieves the desired direction. Furthermore, given a list of points a direction can choose the point that has the minimum relative angle.

To facilitate this use of directions, all directions in iPlot are bounded. That is, both the xy- and z-angles have bounds and the directions are capable of reasoning if the relative angle from any point is within those bounds.

A.5 Dimensions

Dimensions in iPlot are a representation of the size of a physical object. Dimensions are a specification of the width, depth, and height of an object.

Appendix B

Light Specifications & Illumination

Determining the illumination at any point in space from a light requires knowing the photometric properties of the light's beam and the exact placement of the beam in the performance space. iPlot can determine the position of a light beam from its physical model and extract the relevant photometric information from the manufacturer's specifications.

The placement of the light beam is determined by the source and focus points. The source point is the point in space that the lamp, or light bulb, occupies and acts as the source of the light beam. iPlot hangs all lights directly below the hang positions, such that the source point is directly below the hang point, the distance between them is determined by the height of the light. The focus point is the point in space at which the hotspot or center of the lights beam is pointed. The focus line of a light is the line between the source point and the focus point.

The illumination of an area, in other words the amount of light falling on that area by a light measured in foot-candles or lux, is determined by the intensity of light measured in candela or candlepower and the distance from the source point. Areas lit by conventional theatrical lighting instruments have a hotspot, typically at the center of the beam, at which they are brightest and get dimmer as the distance from the hotspot increases following a cosine distribution. This distribution is scaled by knowing the cutoff angle of the light, effectively the angle of the light beam. Theatrical lighting manufacturer provide two angles to specify the beam spread of a light beam:

the beam and field angles. The field angle is defined as the outer cone where light diminishes to 10% of the center intensity. The beam angle is approximately half the field angle and is defined as the internal cone where the light is 50% of the center intensity [10]. The different components of a light beam as specified by manufacturers are shown in Figure B-1.

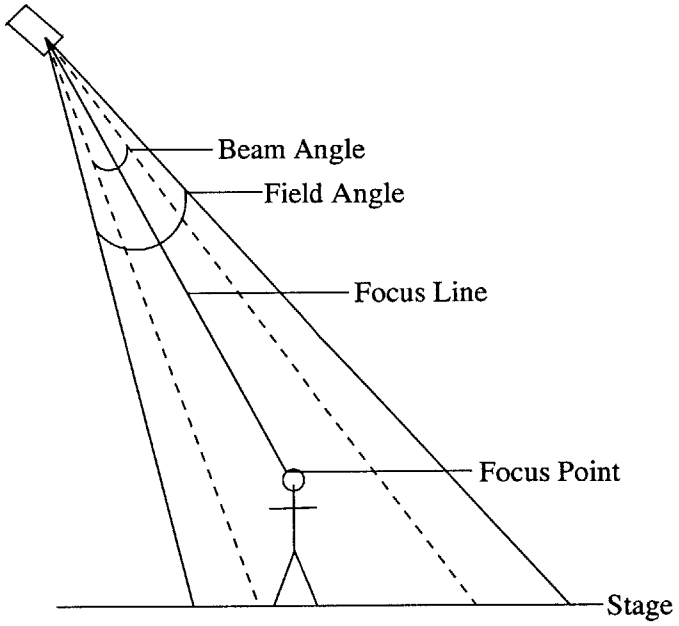


Figure B-1: Light beam description from sepcifications. Entertainment light manufacturers specify the size of the light beam by giving the field angle, where the light diminishes to 10% of the center intensity and the beam angle, where light diminishes to 50% of the center intensity.

From these manufacturer specified angles iPlot can determine the cutoff angle of the light that bounds the size of the light beam. iPlot first assumes that the light beam intensity follows a perfect cosine distribution with the brightest point at the center of the light. Thus we can normalize the field and cutoff angles using a cosine distribution and say $\frac{\text{cutoffAngle}}{\arccos(0)} = \frac{\text{fieldAngle}}{\arccos(.1)}$ and thus

$$\text{cutoffAngle} = \text{fieldAngle} * \frac{\arccos(0)}{0.1}$$

Having assumed the intensity of the light beam follows a cosine distribution and

knowing the intensity at one angle, we can use a similar process to calculate the intensity of the light beam at any angle[1]. To calculate the intensity at any point iPlot finds the angle of that point relative to the focus line. This is one half the angle that determines light beam cone intersecting that point as shown in Figure B-2. Using this cone angle we can find the proportional intensity of the light beam at that point by using the following equation:

$$\text{percentIntensity} = \cos\left(\frac{\text{coneAngle}}{\text{cutoffAngle}} * \frac{\pi}{2}\right)$$

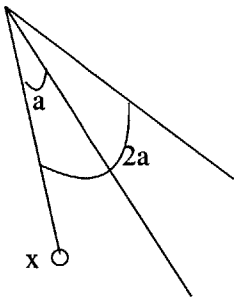


Figure B-2: Calculating percent intensity of a point. Given a point x, we can calculate the angle a between that point and the focus line. Twice that angle is the angle of the cone of the light beam that intersects x. From this cone angle we can calculate the percent intensity at x

Using this knowledge on calculating the percent intensity one can find any point in the light beam at a specified percent intensity. Lights in iPlot have been given a method to find these intensity points. Given an intensity direction from the focus point, this method will find the first point along that line that is at the specified intensity by walking down the points along the line testing each one.

[1]While the cosine function is periodic a light beam is not, so the intensity at any angle greater than the cutoff angle is defined as zero.

Appendix C

Goal Preferences

In order to narrow the search space of possible solutions to a goal the designer can specify a set of preferences that influence the mechanics of finding light and placement pairs and the overall style of the look created by a group of lights. This is done by directly or indirectly specifying the type of lights used to satisfy a goal; the uniformity of illumination level, determined by how lights overlap; the search technique used to find a solution; and, the importance of a single goal in relation to the other goals of the designer. These preferences reduce the search space explored by placing a set of soft constraints on the generators used to create a proposed solution.

C.1 Preferred Light Type

The designer can specify the type of light to be used using two constraints: light type and individual light beam size, also called discretization. In specifying a light type a designer can specify a specific model of instrument, such as an ETC Source Four 26° , which is a ellipsoidal reflector spotlight manufactured by Electric Theater Controls that has a beam spread of approximately 26° . However, in many cases designers have higher level goals, in which they want a specific genre or type of light usually specified by the type of lens or reflector of the light, but not a specific beam spread. Thus designers might specify they want an ellipsoidal, a PAR, or a Fresnel. Given a specified light type of this kind, which is essence a super-type for a wide variety

of different lights with varying beam spreads and intensities, the designer can guide the choice of a specific type of light by specifying a desired amount of discretization, stated in terms of percentage of the width of the area that should be occupied by a single light. Thus, a discretization of 20% would indicate the designer wants it to take five equally sized beams to form a row of lights illuminating the width of the goal area.

C.2 Controlling the mechanics of overlapping lights

In considering the level of illumination in an area, designers constrain how light beams overlap each other and the bounds of the goal area. The crossover intensity preference specifies the point at which two pools of light should overlap, measured in percent intensity relative to the brightest stop of the individual lights. Thus a crossover intensity of 100% would imply that the brightest spot of the lights should overlap (typically the center), which would cause the lights to be directly on top of each other. The standard value for crossover intensity is 50%. At this value the combined pool of light formed by two light beams is as close to a uniform intensity as possible. In a similar vein, the border intensity constraint specifies the minimum percentage intensity at the border, in relation to the hotspot, of a light whose beam overlaps the border area and forms a light at the edge of the light pool.

In addition to these very mechanical preferences determining how lights overlap lighting designers can specify more stylistic preferences that determine the technique and order of laying down multiple lights to satisfy a goal. On the issue of hanging positions, designers must choose if and how they want to prioritize the two angles creating a goal's direction. In finding the "best" hanging position for a light one can consider both angles equally and try to minimize them simultaneously. Alternatively, the designer can choose to prioritize an angle. Thus if the designer specified that he wanted to prioritize the xy-angle, he is in effect saying that he wants the placement that achieves the best xy-angle, and within the multiple hanging positions that achieve this xy-angle choose the one that has a z-angle closest to the goal direction. Designer's

typically prioritize the xy-angle and this is the default for the system.

C.2.1 Determining light placement technique

The order in which lights are placed influences the placement of lights in a stylistic way. While generating in any order will result in the goal being technically satisfied with respect to angle and direction, a designer might still be unhappy with the solution. This situation results because because most large groups of lights by the designer will inevitably be used for multiple purposes throughout a show. As described in Section 2.1 the goal iPlot satisfies may in fact be a combination of several dependent goals. A common example is with a full stage wash, i.e. a group of lights covering the entire stage in light. A designer will often want to turn on only half of these lights, illuminating exactly half of the stage. For this situation to happen the lights must be laid out such that their focus points are symmetric around the centerline.

This type of subtlety of placement is determined by the generation start point and generation technique. The generation start point can be any of eight points: upstage center, upstage right, center stage right, downstage right, downstage center, downstage left, center stage left, and upstage left. Thus a group that's generation start point is upstage left will be generated from stage left to stage right and upstage to downstage. When the start point involves center stage the lights will be symmetric around center. In this case there is a secondary preference called center technique. This preference determines if the group of lights symmetric around center includes a light exactly on center or not. Again this may be useful for points when the designer wants to use a sub-set of the goal's lights.

C.3 Relative goal importance

Because all goals may not be equally important to a designer, each goal is given an order that determines its priority in being satisfied relative to other goals. By default, goals are left in the order in which they are created, but they can be reordered. The

plot generator will generate goals iteratively in order. With a limited inventory goals that are generated first are more likely to be initially satisfied. The goal order may also be considered by a goal repair strategy in choosing repair suggestions.

One preference is considered exclusively by the repair portion of the system. A designer can specify resize directions for a goal. This states the directions, if any, from which the goal area may shrink if there are not enough lights to satisfy the goal. These repair related aspects of goal preferences are discussed further in Chapter 4.

Appendix D

Finding the focus point of a light

When lighting designers place a light they do so in one of two ways, either they know exactly where they want the light to be, i.e. they directly know the focus point, or they know how bright they want the light to be at some point and roughly what direction the center of the light should be relative to that point. When the focus point is not known, the system must search for the correct focus and hang point pair such that the known point is illuminated at the specified intensity. This is done by estimating a first focus point and finding the corresponding hang position for the light, testing to see if it meets the intensity requirements, and if not moving the focus point and repeating the process. iPlot's light position generator has two different methods for searching for a focus point: line bounded search and corner bounded search.

In line bounded search we have a few pieces of knowledge: a line that must have a specific intensity also called the bounding line, a side of the bounding line the focus point must be on, and perpendicular line that must bisect the center of the light beam[1]. As an example of where this comes up in lighting design consider the situation in which the designer placing a light and he wants the bottom of the light pool at 50% the center intensity at the downstage center edge of the stage, as was done in our center generation techniques. In this case the bounding line is the downstage

[1]Because this line is always perpendicular to the bounding line, it can be specified by a single point. Furthermore, the side of the bounding line this single point is on can indicate the side of the bounding line the focus point should be on

edge of the stage, the perpendicular line is the centerline, and the focus point must be above the bounding line because the light is focused above the downstage edge of the stage. The position generator can search for the focus point by choosing points on the perpendicular line, finding a corresponding hanging position, and examining the intensity of a light hung on the hanging position at a specific point, the boundary point. This boundary point is at the intersection of the boundary and perpendicular line, also called the boundary point. If the boundary point is to bright, the focus point must move away from the bounding line, if the boundary point is too dim, the focus point must move closer to the boundary line. This process is demonstrated in Figure D-1.

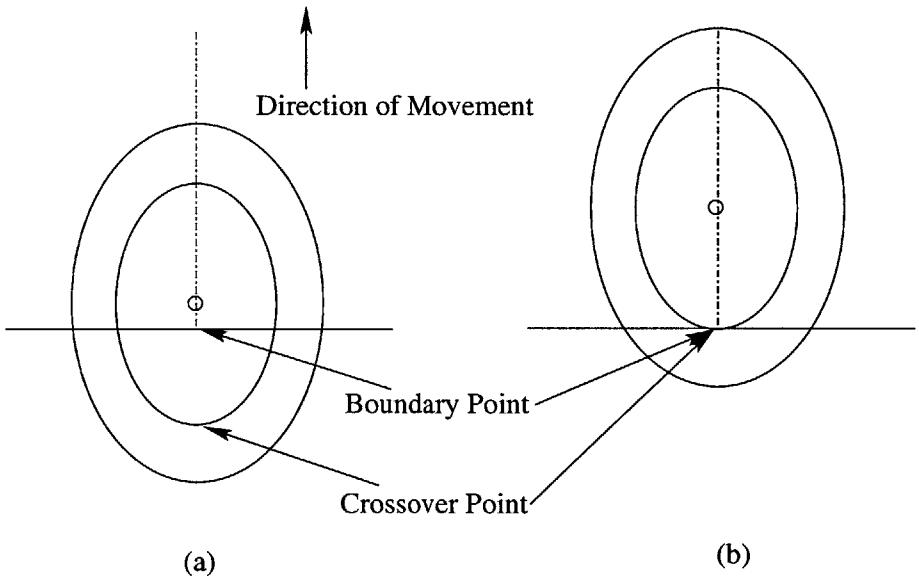


Figure D-1: Finding a light's focus point from a line. A light beam's position is specified a bounding line and a point on that line to be at a specified intensity. The focus point is then found by choosing focus points, testing the intensity at the bounding point, and if necessary moving the focus point such that the intensity at the bounding point approaches the desired intensity.

In corner bounded search we also use two lines, both of which are boundaries: a vertical bounding line and a horizontal bounding line. An example where corner bounded search comes up in lighting design is when lighting a corner of the stage or sub-area, this is the method used by the group generator corner area generation

method. Suppose that a designer wants a light placed such that the 50% intensity light pool just touches the downstage edge of the stage and the centerline, with the light on the stage left half of the stage. In this case the downstage edge of the stage is serving as a vertical bounding line, the intensity at the downstage edge influences the y-coordinate of the focus point. Similarly, the centerline forms a horizontal bounding line and the intensity at the stage left edge of the stage influences the focus points x-coordinate. Corresponding with these lines are a vertical and horizontal bounding points. The vertical bounding point is the point on the vertical bounding line with the same x-coordinate as the focus point. The horizontal bounding point is the point on the horizontal bounding line with the same y-coordinate as the focus point. To find a focus point given the horizontal and vertical bounding lines the position generator picks a point on stage on the correct side of both sides as a focus point and determines the best hang position for that focus point. Given that hang and focus points the generator then determines the intensity of the light at both bounding points. If the intensity of the vertical bounding point is to dim the focus point moves closer to that line, if it is too bright it moves farther away. The same is true with the horizontal bounding point. This process is demonstrated in Figure D-2.

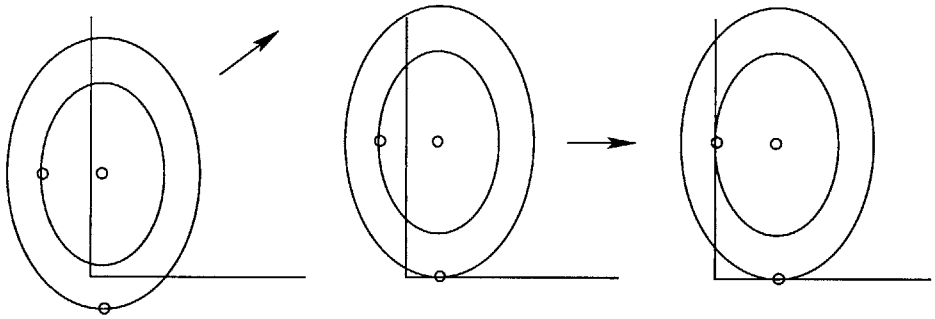


Figure D-2: Finding a light's focus point from a corner. A light beam's position is specified by two bounding lines, each of which must be at a specified intensities. The focus point is then found by choosing focus points, testing the intensity at both points, and if necessary moving the focus point such that the intensity at the bounding points approaches the desired intensity.

Bibliography

[1] Autodesk, San Rafael, California. *AutoCAD User's Guide*, 2005.

[2] Julie O'B. Dorsey, François X. Sillion, and Donald P. Greenberg. Design and simluation of opera lighting and projection effects. *Computer Graphics.*

[3] Electronic Theater Controls, Middletown, Wisconson. *Emphasis Lighting Control System User Manual*, 2002.

[4] John McKernon Software, Phillipsburg, New Jersey. *Beamwright 4 Manual*, 1999.

[5] John McKernon Software, Phillipsburg, New Jersey. *Lightwright 4 Reference Manual*, 2003.

[6] Kimberle Koile. Dependency-directed redesign: Informed search using a mapping of abstract characteristics to physical form. Submitted for publication.

[7] Kimberle Koile. *The Architect's Collaborator: Toward Intelligent Tools for Conceptual Design.* PhD dissertation, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, 2001.

[8] Jason Livingston. Spotlight shines. *Lighting Dimensions*, August 2001.

[9] Stanley R. McCandless. *A Syllabus of Stage Lighting.* Yale University Press, New Haven, Connecticut, 1958.

[10] Steven Louis Shelley. *A Practical Guide to Stage Lighting.* Focal Press, Boston, Massachusetts, 1999.

[11] Reid Simmons. A theory of debugging plans and interpretations. In *Proceedings of the National Conference on Artificial Intelligence*, pages 94–99, St. Paul, MN, 1998.

[12] Michael Sperber. Developing a stage lighting system from scratch. *ICFP*, pages 122–133, 2001.

[13] zBlueSoftware, LLC, Stamford, Connecticut. *LPS Pro User Manual*, 2004.