

Replicating Real World Stage Lighting for AR apps

Gerrit Krot

Department of Computer Science

Golisano College of Computing and Information Sciences

Rochester Institute of Technology

Rochester, NY 14586

gnk1165@g.rit.edu

Abstract—Augmented Reality (AR) applications seek to create realistic experiences that place computer generated objects in physical spaces. Replicating the lighting conditions of these spaces is difficult on wearable AR devices due to their low processing power and lack of lighting data from video input. On many devices, developers are unable to access data from cameras, which means that real-time lighting is impossible. AR-enhanced stage play applications are typically built for a specific stage and lighting setup, and as such, lighting data can be set by developers. While this does allow for increased realism, this process is imprecise, difficult, and time consuming. Additionally, the prevalence of spotlights, which cast light into a very specific area, make indirect lighting techniques like HDR environment mapping ineffective at best. This paper details a method to replicate the lighting conditions present in an image of a lit stage using a genetic algorithm, implemented in an Unreal Engine library. By using a genetic algorithm to optimize the angle, color, and size of spotlights, the lighting environment of a stage can be replicated and used to illuminate digital objects. Currently, this method does not produce results suitable for use, but could be improved with a few modifications.

Index Terms—Augmented Reality; AR; Oculus Rift; Lighting; Stage Lighting; AR-enhanced play; Genetic Algorithm; GA;

I. INTRODUCTION

When someone engages with augmented reality (AR), they expect an immersive experience, where digital objects interact with the physical world. This immersion is predicated on objects interacting with the geometry of this environment as expected: moving across floors, stopping at walls, and being hidden behind nearby objects. Luckily, most AR devices have robust systems in place to map the shape of the surrounding area, allowing developers to have their objects interact with geometry generated on the fly based on the input of cameras on the devices. On mobile devices, these cameras can also be used to approximate lighting conditions present in the scene. There is much research related to improving real-time light and shadow approximations for AR applications on mobile devices.

Unfortunately, many wearable AR devices, (Oculus Rift, HTC Vive, etc.) do not allow applications to access data from the cameras, which significantly limits the ability of the developer to replicate the environment's lighting [1]. Any lighting present in applications built for these devices must be pre-baked into the application. Most AR experiences are built to be used under a wide variety of lighting conditions. For these apps, the only real solution is to choose a suitably

pleasing and generic lighting setup and hope it looks good for most users. Certain applications, however, are designed to be run under specific lighting conditions. These apps can bake in lighting conditions to increase the cohesion between digital and physical objects. One such setting where it would be very desirable to replicate an environment's lighting conditions is in AR-enhanced stage plays. Stage plays typically use a few set lighting conditions and swap between them as needed. Some applications just model all of the lighting conditions virtually [2], but this only really works for simple scenarios. To replicate stage lighting lighting conditions, developers would need to find the position of each spotlight, and then painstakingly figure out their color, angle, and intensity. (Note: Some stage lighting systems do allow for this data to be exported, but many venues do not have this capability). While it may be possible for some of these values to be calculated, this would likely either require specialized equipment, or immense effort from the developers.

A commonly used method for replicating the lighting conditions of a scene is HDR environment mapping. This technique involves taking a few images of a reflective sphere and stitching them together to create a spherical light source around the scene [3]. HDR environment maps are great at modeling diffuse lighting and reflections, but can struggle to model precise details or handle highly directional light sources. For most applications, this is not a problem, but for a stage with lots of spotlights, this is a deal breaker. An HDR environment map captured from the center of a stage would not be able to accurately map the lighting of lights illuminating other parts of the stage. To generate more accurate appearing lighting, this paper aims to model the lights themselves.

This paper proposes a methodology that uses a genetic algorithm to find the color, angle, and intensity of pre-positioned spotlights in a scene. Developers would have to map the geometry of the stage and find the location of the lights, but this data would only need to be taken once and could be re-used for subsequent plays. From there, developers would take an image of the stage under un-lit conditions, and a second image under lit conditions. (Note: This may be changed for the final project). From this, the light value at location on the stage can be calculated. Then, we use the genetic algorithm to generate different configurations of the lights, comparing how they light the stage to our ground truth. Once a sufficiently accurate solution is reached, the lighting configuration is saved, so that

it may be loaded in as necessary. Primarily, this project focuses on creating a pipeline for developers to load stage lighting into Unreal Engine applications for AR-enhanced stage plays. This paper begins by reviewing several other approaches to lighting replication, and the limitations of these methods. Then, we detail the design of the system that optimizes spotlights for stage lighting. After this, we performed a thorough analysis of the results to draw conclusions about shortcomings of this methodology, and present a number of possible improvements and alternate methods.

II. BACKGROUND

There has been much research into various methods to replicate lighting conditions in computer graphics applications and augmented reality.

A. Stage Lighting

Stage lighting offers a unique lighting environment with specific goals and restrictions. The primary goals of stage lighting are to highlight the action, inform the mood, and set the scene [4]. To highlight action, it is important that the characters are well-illuminated and cast clear, defined shadows to make them “pop” off of the stage. This means that lighting generated must be high-contrast, sharp, and directional so as to brightly illuminate and cast clear, dark shadows. Action can be highlighted with character-focused lights that follow the actor’s moves, but motion is out of scope for this project. Informing the mood is generally done by playing off of human perceptions of color, so generated lighting must be color accurate. Finally, some plays use lighting to inform the audience of the location of the scene. Some examples include using different hues and intensities of light to show different times of day, or shining the light through the branches of a tree to make the scene take place in a forest. These effects are possible as long as our generated lighting has crisp shadows and accurate colors, and relevant geometry is precisely modeled. A number of other effects are possible with the use of filters, which can shape the lights or add textures. These effects are out of scope for this project, but could be implemented by uploading an image of the filter used. Since the target application is for AR-enhanced plays, more complicated effects that would generate light are likely to be handled by the computer, and thus would not need to be replicated.

Based on the common conditions of stage lighting, a number of assumptions can be made to refine attempts at replication. While many different lights are used to produce a variety of effects on stage, this paper exclusively focuses on hard-edged spotlights, and lumps all general light sources into a single “ambient” source. While this is done partially for simplicity, it also focuses on replicating the most noticeable part of stage lighting; the sharp regions of light and dark produced by spotlights. Additionally, stage lighting almost exclusively points at the stage, and is reflected to the audience. This means that we do not really need to consider the lighting conditions or geometry of the area outside of the stage. This also means

that lights that do not illuminate the stage do not need to be considered.

B. Lighting Replication Methods

With the assumptions in mind from above, we can look at a number of other approaches to lighting replication to see how they can be applied to this specific problem. Zhao et al. [5] designed a real-time lighting estimation procedure for AR applications on mobile devices. This approach begins by constructing a point cloud that represents the light levels at the surface of each point in the scene. From there, the light direction and diffuse components are estimated using a neural network. This estimation is used to generate an environment map centered around virtual objects. For stage applications on mobile devices, this methodology could likely be applied in concert with modeled spotlights to generate good looking diffuse lighting. For projects designed for wearable devices, this is not feasible, since they do not allow developers to access their cameras.

Gardner et al. [6] create a method for indoor lighting prediction that uses only a single image as input. They start by using a deep neural network to generate a complete 360-degree panorama and an HDR environment map based on the input image. Another neural network is used to detect and classify the light sources in the panorama. Each light is labeled as a spotlight, lamp, window, or reflection. The HDR environment map is then used to estimate the intensities and direction of the located lights. This method is generally very good at creating lighting conditions that look reasonable to other viewers, with no obvious flaws or mistakes. Somanath et al. [3] build on this method by improving the training method to better replicate specular highlights using adversarial loss. Even with these improvements, there are still significant limitations to this approach. First, the network struggles to accurately portray lights that are not in the image. This is to be expected, since it must rely on a prediction of the rest of the scene to predict the location and size of these lights. Secondly, this method is ill-suited to finding the boundaries of sharply lit areas, since it only accounts for light that would strike the simulated object.

This method would likely not be suitable for predicting stage illumination, for a few reasons. Most importantly, the spotlights that we seek to identify can often be stark and sharp, with clearly visible boundaries on the stage. Soft, fuzzy lights are generally not able to cast the sharp, precise shadows that are commonly cast on the stage. Additionally, the step of extrapolating the image to create a panorama would not be needed in this scenario. The bright illuminants would occur from the stage or light sources with known locations. A single image could contain all of the necessary information to replicate the lighting conditions of a stage.

Neural networks and other forms of machine learning are very commonly applied to the problem of lighting replication. They have been shown to produce good results and can be used flexibly. It is likely that a neural network could be designed to position and angle spotlights accurately. The problem with these approaches is that they require a large database of images

associated with accurate lighting replications. No suitable database was found for this project, and so machine learning was not a viable approach.

Another approach for lighting replication was developed by Lopez-Moreno et al. [7]. This method attempts to find the number of lights, their angles, positions, and intensities based on data obtained from an object with known geometry in the scene. First, they map the light information from the pixels to the object, associating each location with an intensity and a surface normal. Using a k-means clustering algorithm, areas of shadow and light are separated. This algorithm also estimates the direction of the lights by checking the relative intensity against the normal vectors. From there, they calculated the color of ambient light from the identified shadow regions. This method generates a number of light sources that illuminate the target object and others in the area. They go on to apply these generated light sources onto a target image to illuminate it in a different way, but the relevant portion to this project is the generated light sources. The lighting generated by this approach is not perfectly accurate, but is more than sufficient for static images. Additionally, the approach functions best when a relatively complex, precisely modeled object is present in the input. This method could likely be applied to stage lighting and could produce solid results, but might require complex geometry to be placed over the entire stage and modeled.

A paper by Madias et al. [8] applied a genetic algorithm to indoor lighting optimization. The authors sought to use the least amount of electricity possible to evenly light a room above a certain threshold. Solutions were evaluated by checking the light level at various points in the room with light sensors. This methodology generated very efficient and uniform lighting of the target levels, effectively replicating the ideal lighting conditions of the author. This project is significantly more complex, requiring the optimization of color, angle, and size of the lights. For an optimization algorithm to work effectively, each solution would need to be evaluated far more precisely. Fortunately, while their testing was bound by the limitations of reality, this project is set in the digital world. What this means is that we can check the accuracy of the light generated by our solution at any number of points in the scene.

Many possible solutions have been tested to replicate the lighting conditions of an environment. Generally, existing solutions are effective at producing conditions with similar levels and colors of light, but struggle when presented with sharp contrasts of light and shadow. Contrast is extremely important to stage lighting, so a new methodology was developed.

III. DESIGN AND IMPLEMENTATION

The goal of this paper is to create a process and tools that allow other developers to replicate stage lighting in their own projects. As such, the project was developed as an Unreal Engine Blueprint library, allowing for native integration into most AR applications. The process used to replicate stage lighting can basically be broken down into the following steps:

1. Model the stage and find the locations of the spotlights.
2. Capture an image of the fully illuminated stage and an additional image for each desired lighting condition (All images must be from the same location at the same angle).
3. Find the light color and real color at each point on the stage based on the images.
4. Use differential evolution to find the optimal angle, color, intensity, and cone-angle of each spotlight
5. Switch between lighting setups as needed by the application. Modeling the stage and capturing images happen outside of the code, and are relatively self-explanatory. For this project, I carefully measured the stage of the Wegmans theater and the distance to the location I captured from, then calculated the angle I took the images at. From the measurements, I was able to reconstruct a rudimentary scale model in blender and import it to Unreal Engine.

A. Calculating colors on the stage

To calculate the level of light at each point on the stage, we can perform a raytracing-like process. We know the field of view of the camera, which tells us the maximum deviation from the base orientation of the camera. This gives us a range of angles that all pixels in the image fall between. Based on this, we can then cast rays out from the camera that pass through the center of each pixel. When these rays hit a piece of stage geometry, we can associate the color information in the image to that location. Right now, all that we can find is the true color of each location, as well as the desired color, so we store those into a structure called a LightPoint that associates the two colors to the location on the stage. The LightPoints generated in this step are used to check the optimality of solutions in the differential evolution step. These colors are stored as 3D vectors based on their RGB values. We must store both values, since we can't determine whether the color of a pixel is due to the light hitting it, or the color of the object. In figure 1 you can see a red ball and a white ball with a white

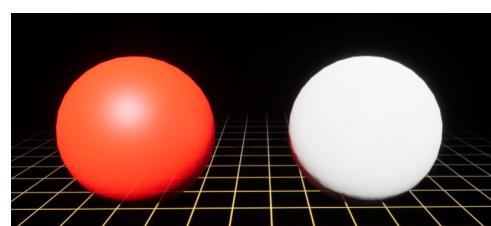


Fig. 1. A red and white ball illuminated by a white light

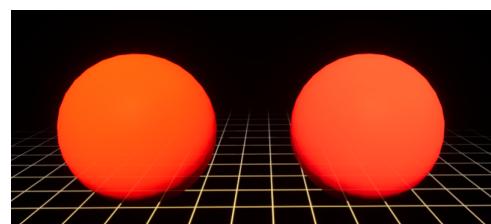


Fig. 2. A red and white ball illuminated by a red light

TABLE I
A VISUALIZATION OF A LIGHT MAP GENERATED FROM TWO INPUT IMAGES



light shining on them, making their true colors obvious. In figure 2, a red light is shown on both balls, making both appear red. When optimizing spotlights, we may find that, instead of shining a red light onto both, a white light is shone on the red ball and a red one on the white ball. To prevent this, we must pass both colors through to future calculations. For this to be useful, we must assume that all parts of the stage are subject to roughly even levels of light in the fully illuminated image. Otherwise, developers would first have to calculate the lighting conditions of their fully illuminated room, which is a harder task than the one solved by this method. In this approach, there could be significant errors if the images do not match up perfectly. Around the edges of objects or shadows, a few pixels could lead to a chunk of the stage having erroneous lighting calculations. As such, it is important to ensure that the images are taken at the same location. Table I shows an image of the target light level at each point. The small offset between the two input images caused by human error results in some minor error in the Lightmap. In general, small errors in this step should not significantly contribute to errors in later steps.

B. Genetic Algorithm

The goal of this step is to optimize the spotlights such that they recreate the desired lighting conditions. To do this, we use a genetic algorithm. Genetic algorithms are a family of optimization techniques based on the process of natural selection which are useful for dealing with problems of high dimensionality [9], [10]. In a genetic algorithm, a population of solutions are created, where each solution contains a list of “genes” that correspond to the variables being modulated in the problem. From there, the solutions that are measured to be the best move on to the next generation. Additional solutions are made via crossover, in which two solutions are combined to produce one new solution. Then, all solutions are randomly mutated, which means that their genes are randomly modified slightly. For this problem, we represent each spotlight with a static 3D vector representing its position and an array of seven floats. These floats represent the location on the stage the spotlight points to, its color, and its outer cone angle, as outlined in Figure 3. Our solution vector is an array of spotlights, meaning that the dimensionality of our problem

is $7n$, where $n = \#spotlights$. There are many variations of genetic algorithms, but the implementation used for this problem is relatively basic.

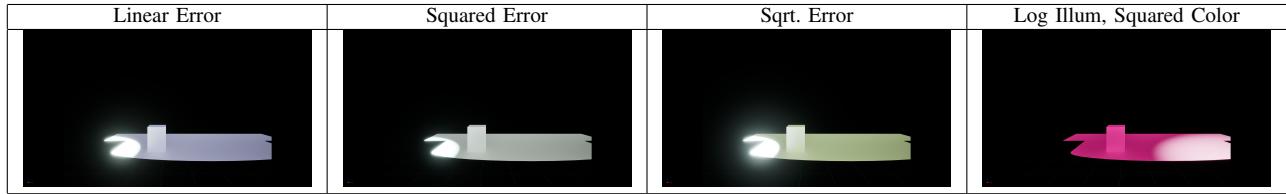
Index	0	1	2	3	4	5	6
Value	Look_At_X	Look_At_Y	Look_At_Z	Color (Red)	Color (Green)	Color (Blue)	Cone_Angle

Fig. 3. The float vector representing a spotlight

1) *Solution Initialization:* First, we calculate the boundaries as defined by our LightPoints. Currently, we operate under the assumption that all spotlights must be pointed somewhere on the modeled stage geometry. While it is definitely possible for lights to be pointing into the audience, or even point at a point just off stage so that they still affect the stage, allowing lights to point where we don't have data would result in a undesired effects. From the programs stand point, there is no difference between a light point that is two feet off stage left and a light shining directly into the audiences eyes. As such, we restrict the lights to pointing onto the stage. Also at this point, we calculate an ambient light, which is the maximum light level possible that does not cause any LightPoints to be a color brighter than desired. At this point, we can randomly generate our starting population. We ensure that all values generated fall in an acceptable range, based on the minimum and maximum sensible values. In addition to the direction, we can also restrict the intensity. There is no way to turn on a spotlight in such a way that it subtracts light from a scene, so the minimum value for each RGB component of color is 0. Currently the maximum is set at two, which means that no individual spotlight is providing more than double the amount of light present on a fully lit stage. This could in theory be inaccurate, but I think that the maximum brightness in most reasonable systems will be very close to that of the fully illuminated stage, capping out around one. Additionally, the cone angle was restricted to a range of $5^\circ - 70^\circ$, which encompasses almost all commercial spotlights[4], and prevents spotlights from being used by the algorithm to reproduce small specular highlights that may have been present in the image.

2) *Selection and Crossover:* Now, we finally begin the optimization process. The first step is selecting the top performing solutions. In this program, we simply select the top 50% of the solution space. There are more advanced methods

TABLE II
A COMPARISON OF TESTED SCORE EVALUATION METHODS
400 GENERATIONS, 50 RESTARTS, POP. SIZE = 60



for this, but performance based selection generally works well enough. The solution vectors then undergo crossover to restore the population to its original size. Random pairs of solutions are selected to undergo crossover. A new solution is created with genes based on the two in the pair. In this program, there is a 40% chance of selecting a gene from either parent vector, and a 20% chance of taking the average of the two. This process helps search the spaces between strong solution vectors, performing more of the “exploiting” part of the “exploring and exploiting” action at the heart of optimization algorithms.

3) *Mutation:* After repopulation, each vector undergoes random mutation. This serves the purpose of exploring more of the solution space. Each individual gene has a small chance to mutate, and can jump a maximum of $\frac{1}{4}$ of the range for that gene in one mutation. In addition, each spotlight in a solution vector has random chance to swap all of it's genes with another spotlight. The reason this is done is to move out of local optima where two spotlights are pointing at spots that the other should be pointing at. By swapping the location they look at, their color, and cone angle, only the shape of the illuminated area is altered slightly, while the genes in the solution appear to have undergone a massive shift. Despite the two solutions appearing similar to outside viewers, the genetic information that produces them are quite different. This mutation procedure allows the algorithm to explore these similar solutions much easier than it would otherwise be able to. After all the genes are mutated, the top performing solution found so far is inserted into the population, assuming it was not already in there.

4) *Using the Results:* This procedure of selection, crossover, and mutation repeats for a number of iterations, and then the top performing solution vector is returned. To ensure that developers do not have to re-optimize their lights every time they run the program, this value is saved to a .json file, which can be loaded to set all of the relevant variables for each spotlight. If a developer was attempting to recreate multiple light settings, all they would need to do is calculate the light map for each pair of images and optimize for each image. They could then load in multiple light settings at the start of a program, allowing them to be swapped between in runtime.

5) *Solution Evaluation:* The most consequential part of this project is the solution scoring portion. The first step is to determine the amount of light from each source that is

illuminating a given light point. This can be done using simple light emission calculations. While it would be ideal to use Unreal Engine's lighting engine for these calculations, this is not possible. Additionally, I could not find exact methodology for how Unreal Engine calculates light, so our calculations do not perfectly reflect the appearance of the lights in the actual scene. They are close enough that this is not a big issue, however. We then use the light effecting that source multiplied by the color present in the base image to calculate a new color. This color is then compared to the target color to generate a score. There are an extremely large number of methods that could be used to compare the two colors. For the purposes of this project, a linear difference was used. Originally, I used the squared difference, or squared error, but this resulted in solutions with reduced contrast. I also tested square root error to see if this would cause the solution to focus more on getting extremely close to targeted illumination values. This may have worked, but resulted in less accurate colors. A final method tested was to calculate the difference in brightness and the difference in color separately. Generally, a small difference in color is significantly easier to detect than a similar difference in brightness. As such, this method added the logarithmic difference in brightness and the linear difference in color. While the idea behind the method was solid, it resulted in terrible colors and terrible brightness.

As you can see in Table II, the differences between each of the methods was minimal, except for logarithmic brightness and linear color. Out of the methods that produced reasonable appearing results, I chose to use the linear difference scoring method. This is because the results appeared to be the best, and because it is the most understandable metric. Constant across all of these methods is that perfectly matching every point gives the best possible score, 0, and lower scores are always better. There are likely countless more methods that could have been used to assign a score to the lighting results, but in the end, I ended up moving forward with linear difference. It is very likely that a better scoring method could have been used, and would have resulted in better results.

IV. RESULTS

The methodology described above was tested on two images captured from the Wegmans Theater at RIT, and a second scenario generated in Unreal Engine. All images input into this method were compressed to 1/8th their original size

TABLE III
A REAL-WORLD TEST CASE OF RIT'S WEGMANS THEATER, OPTIMIZED BY OUR GENETIC ALGORITHM METHOD
400 GENERATIONS, 50 RESTARTS, POP. SIZE = 60. TIME TO RUN = 30 MINUTES

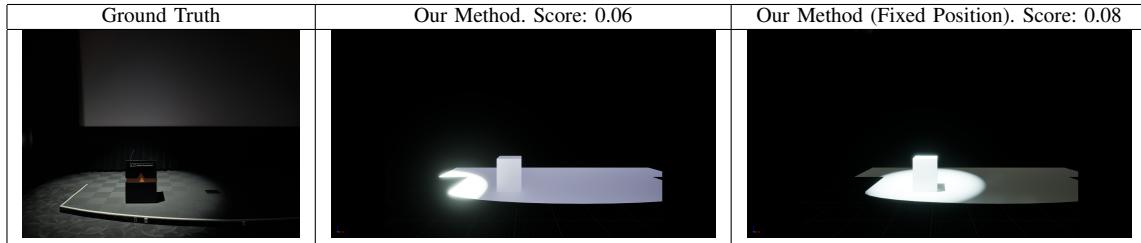
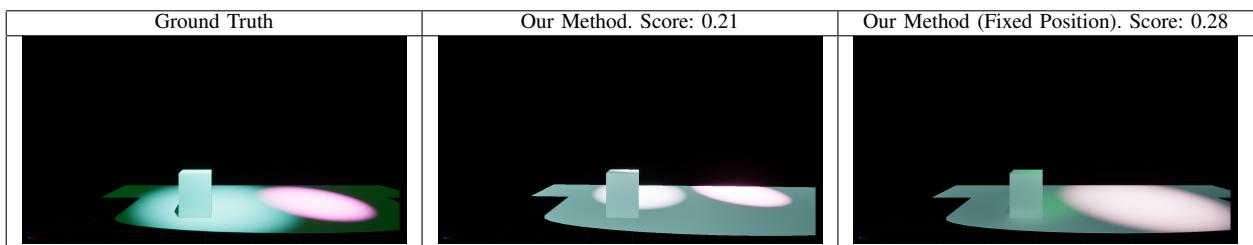


TABLE IV
A SIMULATED TEST CASE PRODUCED IN UNREAL ENGINE FEATURING 2 SPOTLIGHTS.
400 GENERATIONS, 50 RESTARTS, POP. SIZE = 200. TIME TO RUN = 3 HOURS



to improve the speed of the algorithm.

This algorithm can produce results that are fairly close in terms of color and intensity, but struggles to have them cover the entire lit area. This is due to the scoring procedure, which appears to incentivize spotlights shining only on the brightest parts of the image and increasing the brightness of the ambient light to compensate. The scoring procedure does not appear to represent reality as well as ideal, results that look significantly worse to the human eye can score much better. As you can see in Table III, the test with fixed location appears far closer to the ground truth, yet has an error score 0.02 higher than the fully optimized test. It is also possible that this discrepancy was caused by the image capture methods not accounting properly for specular highlights or reflected light. That being said, the dull gray carpet of the theater should not have produced bright highlights. When the angles of the spotlights are fixed, the genetic algorithm produces results that appear very accurate in terms of color, spotlight size, and intensity since it is more difficult to overfit to the brightest parts. Additional tests were run while fixing the outer angle of the spotlight, but this did not improve results in this case.

The multi-light tests in unreal engine further show the tendency of the algorithm to overfit to the bright parts. Examining Table IV shows the two spotlights covering a smaller range than the target image and the ambient color being made brighter to compensate. While the camera matchup was far more accurate for this test, it is possible that the bright reflective stage surface increased the error due to specular highlights. Interestingly enough, the fixed location test looks less accurate, despite ensuring that the spotlights were fixed at

the correct locations. It is possible that increasing the number of generations would have helped improve the results, but the fixed location test showing worse results would suggest otherwise.

This method is very computationally expensive, taking multiplicatively longer based on the number of runs, generations, population size, input image size, and number of lights in the scene. Since each light added also adds another seven dimensions to the solution space, the time to converge increases dramatically with additional lights. This means that the number of runs, generations, and population size must be increased to allow the solution to converge, further increasing the run time of the algorithm. In addition to the two test cases above, a six spotlight case was attempted, but scrapped due to the multi-day convergence time. In most cases, a developer will be able to estimate a reasonably accurate solution far, far faster than this algorithm can.

V. CONCLUSION

Overall, this method was unable to produce satisfactory results unless under significant constraints. I think that the inaccurate results could be fixed by improving the solution evaluation method. The algorithm did not seem to be getting stuck in local optima too often, and was able to produce very similar results when ran multiple times in a row, so the genetic algorithm portion of the project is performing well. The optimization algorithm generates solutions with relatively low error scores, but the scoring function does not represent what we want it to. Using an optimization algorithm to replicate stage lighting shows some amount of promise, and has much more room for exploration. This implementation

is also far too slow to be really useful. The process of measuring the stage and capturing images of the lit scene already requires developers to have physical access to the environment. Developers would be able to manually measure and recreate the lighting of their target scene faster and more accurately than this approach. For now, developers looking to replicate stage lighting in their project should take careful measurements and attempt to manually replicate the lighting.

VI. FUTURE WORK

There is much future research to be done on this topic, both in terms of improvements to this method, and other methods that may produce better results.

A. Improvements

The most obvious way to get better results out of this project would be to use a more advanced solution scoring algorithm. While it is easy to verify a solution as correct based on the light mapping data, it is far more difficult to decide whether one solution is better than another. This project only tested a few simple variations on the difference between the light levels at each point and was unable to produce a usable replication of lighting. There are many metrics and analytics that could be used to better score a solution. The speed of solution evaluation also could have been dramatically improved by utilizing the GPU. The algorithm is structured in a way that would allow for the entire step of solution evaluation to be parallelized, which could allow for results to be generated in a matter of minutes, not hours.

Another improvement would be to the light mapping algorithm to reduce noise and variation due to different surfaces. While the approach used by this project accounted for the different color of materials, it still produced results with a lot of noise and variance between pixels that should be receiving the same light value. Some of this could be solved by using better capture techniques and ensuring that the two images were captured from exactly the same place. Some pre-processing steps may also be helpful. For each test in this project, I was compressing the input images to speed up the results. This effectively acted as a method of sampling the image to reduce the amount of light level checks at nearly identical locations. It is likely that a more clever sampling method could be used to dramatically reduce the number of tracked light points without sacrificing any information. In addition, using information from neighboring pixels could also allow for smoother images that more accurately represent the human perception of the scene.

It may also be possible to gain additional information from the evaluation calculation to better hone the solution. Currently, each solution is mutated randomly, but we could calculate the manor of error in each spotlight and mutate related to that. For example, one spotlight might not add enough light to any of the points it affects, so we could increase the odds of mutations that boost its luminance. Another spotlight might be too bright for points only on the left side, so we would emphasize mutations that point it further to the

right. This would likely increase the speed of convergence, but could lead to the algorithm getting stuck in local optima.

B. Other Methodologies

Another relatively simple approach to replicating stage lighting could make use of the fact that most spotlights project their light onto the stage in the form of an ellipse. By using a circle or ellipse detection algorithm on the initial image, the boundaries of each lit area could be found. From there, it might be possible to calculate the outer angle and direction of each spotlight based on the shape of the ellipse, and the color and intensity based on the pixels contained therein.

Another approach might be to use a machine learning algorithm. By constructing a database of illuminated stages and their lighting conditions, it may be possible to train an algorithm to generate ideal results. Multiple previous papers have had success using machine learning to generate HDR environment maps [6], [3] or otherwise refine estimated lighting conditions. It would not be surprising if a machine learning approach was able to efficiently replicate stage lighting as well.

ACKNOWLEDGMENT

I would like to thank Dr. Joe Geigel for his help and guidance throughout the entire process of developing this project. His insights and ideas were valuable in formalizing the idea and methodology that allowed this project to come together.

REFERENCES

- [1] Meta. (2024, Sep.) Passthrough api overview. Website. [Online]. Available: <https://developers.meta.com/horizon/documentation/unreal/unreal-passthrough-overview/>
- [2] N. Sugano, H. Kato, and K. Tachibana, “The effects of shadow representation of virtual objects in augmented reality,” in *The Second IEEE and ACM International Symposium on Mixed and Augmented Reality, 2003. Proceedings.*, 2003, pp. 76–83.
- [3] G. Somanath and D. Kurz, “Hdr environment map estimation for real-time augmented reality,” in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, jun 2021, pp. 11293–11301. [Online]. Available: <https://doi.ieeecomputersociety.org/10.1109/CVPR46437.2021.01114>
- [4] K. Stamp. (2023, Aug.) Stage lighting design, part 2: Objectives of lighting design. Blog. [Online]. Available: <https://blog.etcconnect.com/stage-lighting-design-part-2-objectives-of-lighting-design>
- [5] Y. Zhao and T. Guo, “Pointar: Efficient lighting estimation for mobile augmented reality,” in *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIII*. Berlin, Heidelberg: Springer-Verlag, 2020, p. 678–693. [Online]. Available: https://doi.org/10.1007/978-3-030-58592-1_40
- [6] M.-A. Gardner, K. Sunkavalli, E. Yumer, X. Shen, E. Gambaretto, C. Gagné, and J.-F. Lalonde, “Learning to predict indoor illumination from a single image.” *ACM Trans. Graph.*, vol. 36, no. 6, Nov. 2017. [Online]. Available: <https://doi.org/10.1145/3130800.3130891>
- [7] J. Lopez-Moreno, S. Hadap, E. Reinhard, and D. Gutierrez, “Compositing images through light source detection,” *Computers Graphics*, vol. 34, no. 6, pp. 698–707, 2010, graphics for Serious Games Computer Graphics in Spain: a Selection of Papers from CEIG 2009 Selected Papers from the SIGGRAPH Asia Education Program. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S007849310001299>
- [8] E.-N. D. Madias, P. A. Kontaxis, and F. V. Topalis, “Application of multi-objective genetic algorithms to interior lighting optimization,” *Energy and Buildings*, vol. 125, pp. 66–74, 2016. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0378778816303553>

- [9] Bilal, M. Pant, H. Zaheer, L. Garcia-Hernandez, and A. Abraham, "Differential evolution: A review of more than two decades of research," *Engineering Applications of Artificial Intelligence*, vol. 90, p. 103479, 2020. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S095219762030004X>
- [10] D. Fogel, "An introduction to simulated evolutionary optimization," *IEEE Transactions on Neural Networks*, vol. 5, no. 1, pp. 3–14, Jan 1994.