Projector Systems to Control the Material Perception of an Object

Robert van Dijk
Supervisor(s): Elmar Eisemann, Michael Weinmann, Baran Usta
EEMCS, Delft University of Technology, The Netherlands

June 19, 2022

A Dissertation Submitted to EEMCS faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering

Projector Systems to Control the Material Perception of an Object

ROBERT VAN DIJK

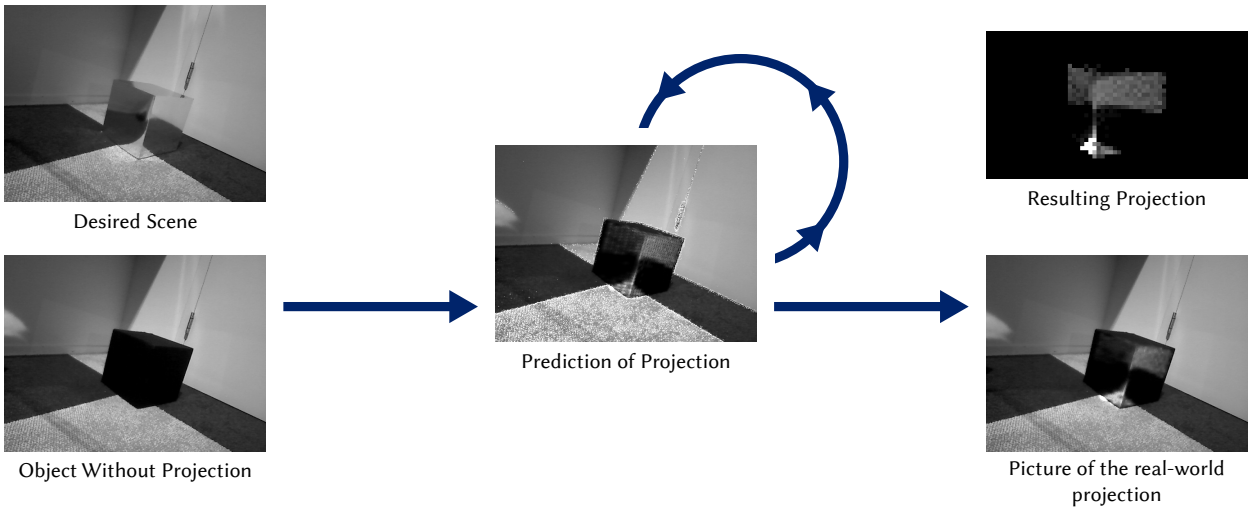


Fig. 1. Results in the real world. The desired scene has a mirror-like cube. This is then replaced with a black fabric cube. A calibration is done. Then an initial guess for a projection gets optimized by minimizing the difference between the desired scene and the prediction of the outcome. This projection is then projected onto the cube.

The appearance of an object or scene is determined by factors like the material, the lights, the geometry, the position of the observer, and the surroundings. Changes in these factors can be simulated using a projector-camera setup. Other research focuses on changing the appearance from the perspective of the projector, or on projector compensation for slightly warped planar surfaces. This paper aims to simulate changes in the scene's appearance by actively manipulating the lighting using a projector-camera setup. It works on not only planar surfaces, but also on more complex geometries. This is achieved by first doing a calibration, and then using this to optimize a projection. This projection is optimized to minimize the difference between how the scene looks when the projection is projected and the desired scene. For low-resolution projectors, it can do this in a few seconds to half a minute. For higher resolutions, the calibration time and file size get quite big. This can be solved in future work using different calibration methods.

1 INTRODUCTION

The appearance of an object or scene is determined by factors like the material, the lights, the geometry, the position of the observer, and the surroundings. This paper aims to simulate changes in the scene's appearance by actively manipulating the lighting using a projector-camera setup. This technique can be used for a multitude of projection mapping applications, like Augmented Reality experiences in theme parks [12], or advertisements on buildings and other objects [9].

June 19, 2022.
Responsible Professor: Elmar Eisemann.
Supervisor: Michael Weinmann.
Supervisor: Baran Usta.
Author's address: Robert van Dijk.

Changing the appearance of an object or scene is already possible using approaches like the one in the work by Amano [1] or Fujii et al. [4], where they used a coaxial projector-camera system to change certain material properties. Another approach to change the appearance is by creating a desired scene, and then warping and transforming this by hand until the desired projection is reached, which is then projected onto the object.

Current research either focuses on changing the material from the perspective of the projector, or doing projection mapping of another image, but not much research on the combination of this has been conducted. This paper proposes a technique to give an arbitrary object a different appearance, from a perspective different that can be different from the position of the projector. An example is shown in Figure 1. Here, a projection is projected onto a diffuse black cube, to make it look like a mirror-like cube.

Section 3.1 shows a first initial prototype. This prototype is used to identify limitations, and is improved upon in Sections 3.2-3.4. After that, the results and performance of these methods are shown in Section 5. These are discussed in Section 6.

2 RELATED WORK

Amano [1], and Fujii et al. [4] change the appearance of certain material properties by using a coaxial projector-camera system. This way it is scene-independent, but as observer position, they only considered the perspective of the projector. The approach in this paper considers a perspective different from the projector's position.

Huang and Ling [7] presents an end-to-end solution for projector compensation. Here they compensate for the color and warping of

the observed image. But this technique is developed only for planar surfaces. Our approach is not limited to planar surfaces, but also allows projections on general geometry.

The work by Huang and Ling [6] is an extension to the previously mentioned paper. They introduce a geometric correction subnet that compensates for a more deformed projection surface. But in their paper, they fail to show the performance on a surface that is more irregular than a slightly deformed planar surface.

The research by Anrys et al. [2] changes lighting to make a scene appear to the observer in a particular way. This is done by first recording basis images for every light source, and then optimizing the intensity of the light source to get the desired view. This is similar to what is presented in this paper, but a major difference is that instead of lamps as light sources, we use a projector for manipulating the lighting in the scene. Then, every pixel can be considered a light source, which is often more light sources than the 40 lamps from the research by Anrys et al..

3 METHOD

To manipulate an object's or scene's appearance, we use a projector-camera setup, of which an example is shown in Figure 2. The camera is placed at a different angle than the projector. By doing this, the projector-camera setup can be used as a stereoscopic setup. This way more data about geometry can be gathered, if necessary. This setup can be built in the real world, or virtually for testing.

In the scope of this work, we tested multiple approaches to changing an object's, or scene's, appearance. Each approach had its own limitations, and the next approaches tried to improve on that.

First, in Section 3.1, a naive prototype is made to show some limitations. Then, the limitation of having to know the geometry of the object to project on will be investigated. This will be tackled by using an optimizer in Section 3.2. To speed this up, calibration will be performed in Section 3.3. A calibration comes with the advantage of knowing the gradient, which is used in Section 3.4.

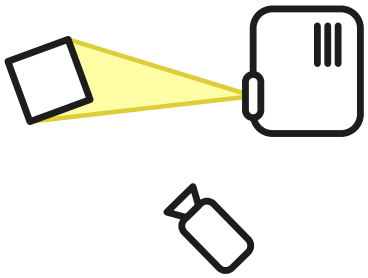


Fig. 2. Projector-Camera setup. Here the camera is off-axis from the projector.

3.1 Naive prototype

The naive prototype works as follows (see the diagram in Figure 3).

First, one creates a 'desired scene'. In this scene, one places some geometry and gives it a material of their choice. One could also add virtual lights to the scene in different colors.

If the projection should look natural in the real-world scene, the real geometry, textures, materials, and lights of the surroundings also have to be modeled. This way metallic and specular reflections end up in the correct place.

This desired scene is observed by a virtual camera. This virtual camera needs to be positioned in the same place the observer in the real world would be. The output of this virtual camera is then re-rendered to generate the image that can be projected using the real-world projector. This is done by first projecting the image from the virtual camera on the geometry of the real-world object from the perspective of the observer. This geometry is then rendered from the perspective of the projector with another virtual camera.

The final image from the second virtual camera is sent to the real-world projector and projected onto the real-world object.

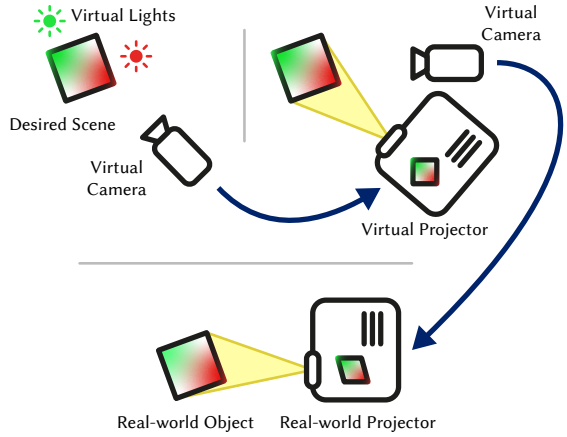


Fig. 3. Pipeline of the naive prototype. A virtual desired scene is created. This is captured by a virtual camera and projected from a different perspective back on the object. This is again captured by a virtual camera, and that is projected on the real-world object.

3.1.1 Limitations. Results can be seen in Figure 4. In Figure 4a, the desired scene from the observer's perspective is shown. This is a cube, with a light pink metallic surface. In this scene, there are also two virtual lights, one red and one green.

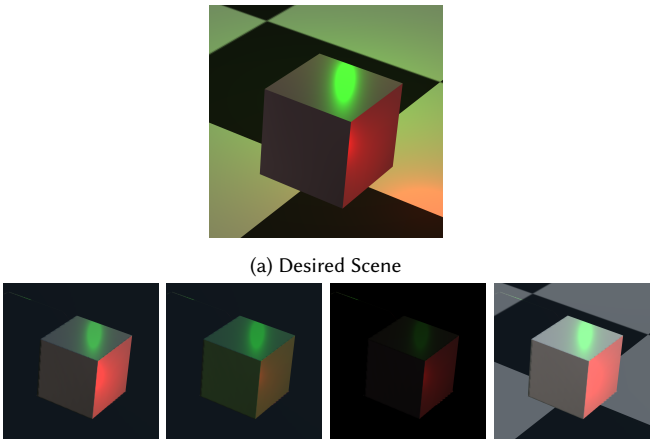
In Figure 4b, the result of the whole pipeline and simulation is shown. In this scene, the object that is projected on is a cube, with a white diffuse material. This scene also has no additional lights besides the projector. It can be observed, that the green reflection is not as bright as the green reflection in the desired scene. Next to this, the red reflection appears way brighter than the one in the desired scene. This can be explained by the fact that for the green spot, the angle of incidence of the projector's light is big, so less light is reflected. The light for the red spot is projected with a low angle of incidence, so it appears brighter.

In Figure 4c, the same projection is shown, but this time it is projected on a green surface. Here, the red is almost not visible, and the gray parts also appear green.

In Figure 4d, the same projection is shown again but now projected on a very reflective surface. This surface almost behaves like

a mirror. Here, the reflection almost does not appear, since the light rays from the projector are perfectly reflected. Another problem is that the observer sees the rest of the room in the cube.

Lastly, Figure 4e shows the same projection on a white diffuse object, but in a bright room. Here, there are also other light sources that illuminate the object. The darker parts of the projection appear brighter since those parts are also illuminated by the ambient light.



(a) Desired Scene

(b) Perfect conditions (c) Green surface (d) Reflective surface (e) Bright room

Fig. 4. Limitations of the naive prototype. The desired scene goes through the pipeline of the naive prototype and is then projected onto cubes of different materials or in different lighting.

3.2 Using an optimizer to obtain the projection

To overcome the limitations of having to know the geometry of the object, we incorporated an optimizer to optimize the to-be-projected image, to minimize the difference between the desired scene and actual scene. This approach, as can be seen in Figure 5, is an iterative process.

In the first step, an initial guess is done to what the projection is supposed to be. This could be random, or an optimized guess. Then, this initial guess is projected onto the object and observed by the camera. This happens in the real world using a real projector, or in a simulation using a virtual projector. Now the mean squared error (MSE) is computed between the image observed by the camera and the desired appearance. The projection is then updated and the cycle starts again.

By updating the projection, the MSE should be minimized. More formally, we want to minimize $O(P)$ in the following equation:

$$O(P) = \frac{1}{n}\sum_{i=1}^{n}(f(P)_i - D_i)^2 \tag{1}$$

Where P denotes the projected image, $f(P)$ denotes how the projection appears to the observer, D denotes the desired view from the observer and n denotes the number of pixels in the camera. In this second prototype, $f(P)$ is acquired by projecting the projection on the object and capturing it with a camera.

Using an optimizer, with a variable for every projector pixel, a new projection can be calculated. L-BFGS-B can be used for this since it is suited for problems where the gradient is hard to find [14].

Using the projector-camera pipeline is very slow since in every iteration an image has to be sent to the projector, and the projector has to project that image. Then the manipulated scene has to be observed by the camera and captured on the computer. There had to be a 0.1-second delay between sending the projection and capturing the image. This can be overcome by making a virtual model of the scene and projector, using a calibration.

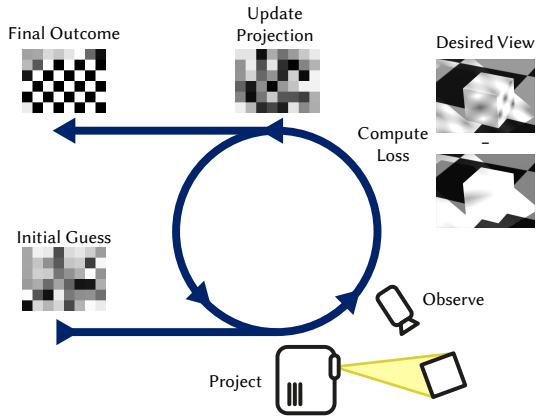


Fig. 5. Pipeline of the iterative prototype. An initial guess is projected and observed. The loss is computed, and an optimizer updates the projection. The cycle repeats until some stop condition is reached.

3.3 Measuring the per-pixel influence of the projector

To speed up the projection mapping pipeline, a model of the scene and projector can be built using an initial calibration. This calibration can be used to estimate what the scene would look like if a projection would be projected onto the scene.

The calibration starts with capturing the scene with all projector pixels off. This image shows the scene with just the ambient lighting. After that, it lights up all the pixels one by one and captures the images. It subtracts the image with all the pixels off from the images with one pixel on. This can be seen in Figure 6.

Fig. 6. The calibration step for 3 of the projector's pixels. For every pixel in the projector, a picture is taken. The picture where all projector pixels are off is subtracted from these pictures. The results are stored in the calibration matrix.

When calibration is done, the calibration matrices are stored according to

$$A = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} \end{bmatrix} \qquad B = \begin{bmatrix} b_1 & \dots & b_n \end{bmatrix} \tag{2}$$

Here A represents the images taken for every projector pixel, where every row is one camera image with n pixels. There are m pixels in the projector, and in every row, a different pixel was lit up. B denotes the picture with all pixels off and is just one image with n pixels.

Using matrices A and B, one can calculate how the projection would look without projecting it. Let this function be denoted by g. Then $g(P)$ would be defined according to:

$$g(P) = P \cdot A + B \tag{3}$$

Using this model drastically speeds up the optimization since we can compute Equation 4 without waiting for the projector-camera pipeline:

$$O(P) = \frac{1}{n} \sum_{i=1}^{n} ((P \cdot A + B)_i - D_i)^2 \tag{4}$$

3.4 Using the gradient

In the prototype from the previous section, the optimizer still computes the gradient by doing a numerical approximation of the Jacobian matrix. When supplying the gradient to the optimizer, it can speed up drastically, like Anrys et al. found in their paper [2]. Since there is now a simple equation for finding the projection, we can calculate the gradient for the optimizer. This is done automatically by a library like PyTorch [13]. The respective pipeline is illustrated in Figure 7.

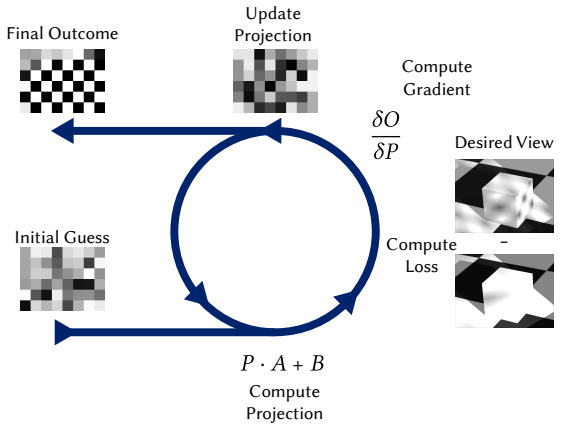


Fig. 7. Pipeline of the gradient prototype. An initial guess is multiplied by the calibration matrix. The loss and the gradient are computed. An optimizer updates the projection. The cycle repeats until some stop condition is reached.

Now an optimizer that needs the gradient can be used. Here we used the Adam optimizer. This optimizer is suited for problems with many parameters, and noisy or sparse gradients [8]. This is useful in our situation since an image can be very big, and not all pixels have a good gradient since some pixels fall outside of view.

4 IMPLEMENTATION

This section highlights the different technologies used to implement the simulation and the prototypes. The simulation was built in Unity, because of the wide range of libraries and documentation for real-time visualizations. Python was used for calibration and optimization, because of the wide range of data processing and machine learning libraries.

4.1 Simulation

In Unity, a simulation of the real-world part of the project has been developed. This was helpful since here we can control all variables like material and ambient lighting. It also provides cameras with no noise, that stand perfectly still during the experiment. This reduces errors in the calibrations, which is useful for early experiments.

A scene with a room and possible ceiling lights was modeled. In this scene, a virtual projector, and a virtual camera were placed. For this part, the Projector Simulator[1] by White Games was used.

To pipe the video streams from Python to the simulation, and back, Syphon[2] was used. In Unity, the KlakSyphon[3] library by GitHub user keijiro was used. And in Python, the syphonpy[4] wrapper by GitHub user njazz was used.

4.2 Naive prototype

This prototype was fully built in Unity. For the step where the perspective of the image is changed from the observer to the projector, the unity-camera-projection[5] library by GitHub user komietty was used.

4.3 Iterative prototype

The iterative prototype uses the SciPy implementation of L-BFGS-B[6]. When not using the simulation, it uses OpenCV to capture an image from the camera. It uses NumPy to calculate the MSE between the image received from the camera or simulation, and the desired view.

4.4 Calibration

The calibration uses Syphon in the simulation, or OpenCV to capture the images. The captured frames are stored in NumPy arrays. The NumPy arrays are subtracted and clamped between 0 and 255. These arrays are then stored in the NumPy format on the disk.

4.5 Gradient prototype

The gradient prototype uses PyTorch to optimize the image. The PyTorch model has a parameter for every pixel in the projected image. Then, on the forward call, it uses the calibration matrices as

[1] https://assetstore.unity.com/packages/tools/particles-effects/projector-simulator-86123
[2] http://syphon.v002.info
[3] https://github.com/keijiro/KlakSyphon
[4] https://github.com/njazz/syphonpy
[5] https://github.com/komietty/unity-camera-projection
[6] https://docs.scipy.org/doc/scipy/reference/optimize.minimize-lbfgsb.html

PyTorch Tensors to calculate the predicted projection. It does that using PyTorch's implementation of the dot product, addition, and clamping.

By using PyTorch's functions and Tensors, PyTorch remembers exactly what calculations were done in the forward step and can automatically calculate the gradient using PyTorch's automatic differentiation [13].

5 RESULTS

This section shows the results of different prototypes. All results were achieved on a 2,3 GHz 8-Core laptop, with 16 GB of memory. The GPU in this laptop is a Radeon Pro 560X with 4 GB of memory. When there was a simulation used, this simulation simultaneously ran on the same machine.

5.1 Duration of the calibration

The calibration can be really slow since there is a delay between sending the image to the projector and being able to capture that projection using the camera. Therefore, it is interesting to measure how long the calibration takes.

This experiment measured the time it took from the start to the end of the calibration, not counting the time it took to write the calibration file to the disk. The experiment was conducted using the simulation. The results for different projector and camera sizes are shown in Table 1. In Table 1, there are also some results by linearly extrapolating results with a lower projector resolution. These are highlighted in grey.

Table 1. The time it took for the calibration to run for different projector and camera resolutions. Grey results are linearly extrapolated from the results with a lower projector resolution and the same camera resolution. Note the changing units.

Projector resolution	Camera resolution			
	16×9	640×360	854×480	1920×1080
4×3	3 s	3 s	3 s	4 s
40×30	268 s	273 s	282 s	365 s
80×60	1085 s	1123 s	1127 s	1447 s
1920×1080	127 h	135 h	135 h	173 h

5.2 Size of the calibration

Since the calibration stores a full image of the scene for every pixel in the projector, the calibration file could get very large. Especially since the current implementation stores uncompressed images in the form of NumPy arrays on the disk.

Table 2 shows the same setups as in the previous section, but now shows the size of the calibration files. In Table 2, there are also some results by linearly extrapolating results with a lower projector resolution. These are highlighted in grey.

These calibration files could be compressed when stored on the disk. When compressing the 10 GB file from Table 2 using zip, the resulting file is only 14.2 MB.

Table 2. The size of the calibration files for different projector and camera resolutions. Grey results are linearly extrapolated from the results with a lower projector resolution and the same camera resolution. Note the changing units.

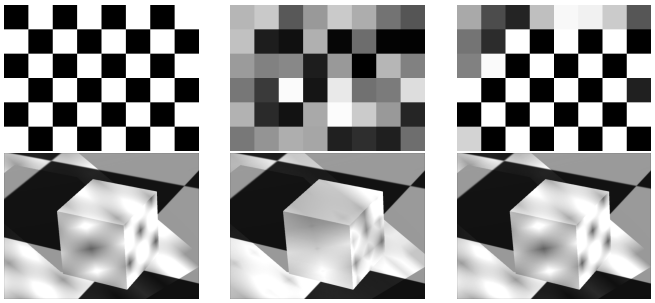
Projector resolution	Camera resolution			
	16×9	640×360	854×480	1920×1080
4×3	2 KB	3 MB	5 MB	27 MB
40×30	173 KB	276 MB	492 MB	2.5 GB
80×60	692 KB	1.1 GB	1.9 GB	10 GB
1920×1080	30 MB	476 GB	818 GB	4.3 TB

5.3 Performance without gradient

Here, the prototype using the L-BFGS-B algorithm was run to create the projection for the desired image in Figure 8a. This desired image was generated by projecting a checkered image onto the scene so that the desired scene was actually achievable.

This experiment was conducted using the calibration model of an 8×6 px projector, and a 640×480 px camera. It uses the default stopping conditions of the SciPy implementation of L-BFGS-B[7]. It used an absolute step size of 0.1 to numerically approximate the gradient. The algorithm was run 10 times, and the average run times and iterations were recorded.

Every instance started with a random initial guess for the projection, one of which is shown in Figures 8b. Then, after an average of 12 seconds and 1225 iterations, one of the stop conditions was reached. The resulting image of one of the runs is shown in Figure 8c.



(a) Desired scene generated using the projection above

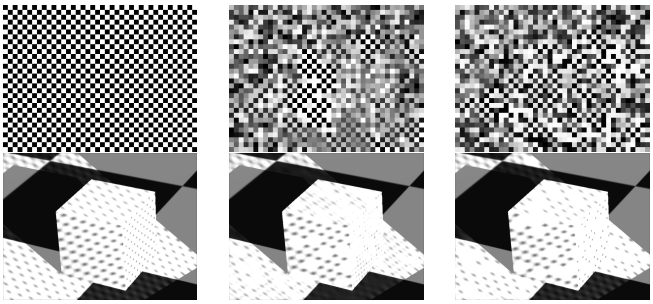
(b) Scene with the initial projection

(c) Scene with the resulting projection

Fig. 8. Results of the iterative prototype. An 8×6 px projector was used. The prototype started with a random initial guess, and after an average of 1225 iterations, ended up with a picture that resembles the desired scene.

The same experiment was conducted at a higher resolution, namely a projector resolution of 80×60 px. The optimizer stopped after 15613 iterations, which took 46.5 minutes because the number of function evaluations reached its limit. The resulting images can be seen in Figure 9b.

[7] https://docs.scipy.org/doc/scipy/reference/optimize.minimize-lbfgsb.html

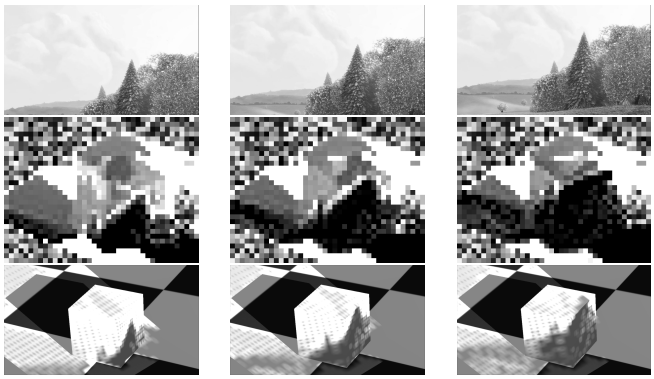


(a) Desired scene generated using the projection above

(b) Scene with the resulting projection without using the gradient

(c) Scene with the resulting projection using the gradient

Fig. 9. Results of the iterative prototype, and the gradient prototype on higher resolution. A 80×60 px projector was used. The iterative prototype produced this in 46.5 minutes, and the gradient prototype in 18 seconds.



(a) Frame 10 (b) Frame 11 (c) Frame 12

Fig. 10. Using a video as the desired image. The top image is the video frame, the middle image is the generated projection, and the bottom image is the predicted result. Since a video often has temporal continuity, the projection between frames only has to change a bit. Therefore, the generated projection of the previous frame is a good initial guess for the next one.

5.4 Performance using the gradient

In this experiment, the prototype using PyTorch was run. Here, the Adam optimizer was used with a learning rate of 0.05. It uses the same calibration model as the experiment from Section 5.3. The results look indistinguishable from Figure 8.

The run time, however, is drastically lower. It took, on an average of 10 runs, in 0.30 s and 34 iterations.

It took for the higher resolution version, from Figure 9, on average, 18 seconds and 114 iterations.

5.5 A video as the desired image

Here, a video was used as the desired images. This experiment used the prototype using PyTorch. It used a calibration with an 854×480 px camera, and a 40×30 px projector.

The 10:35 min video, Big Buck Bunny[8] was converted to an 854×480 px resolution, and 1 fps frame rate. This was then played back where each frame was a new model that needed to be optimized. As an initial guess, the result of the last frame was used. This is a good initial guess, as most videos have high temporal continuity [5], which means that only a few pixels have to be changed between frames. A few frames are shown in Figure 10.

With the stopping condition of the loss decreasing by less than $1 \cdot 10^{-5}$, the video played in 46 minutes, which is 4.4 seconds per frame. In contrast, when using a random initial image for each frame, the video played in 82 minutes, which is 7.8 seconds per frame.

5.6 Real-world results

To test if the technique would work in the real world, a test setup was created as can be seen in Figure 11. For the webcam, the Philips SPC1030NC[9] was used, and for the projector, the InFocus IN3118HD[10] was used. As webcam resolution 640×480 px was used, and as projector resolution 64×36 px was used, The object was a 20×20 cm laser-cut wooden cube, with a replaceable cover. Here the cover was mirror-like paper, white paper, or black paper covered in fabric.

Fig. 11. Setup of real-world experiments.

In this experiment, a cube with a mirror-like material was photographed in the scene, as the desired scene (see Figure 1). After this, a calibration was performed on a cube with a black fabric material. Because of the noise in the camera, the calibration had a lot of errors. This was resolved by, in this experiment, setting every value below 10 (out of 255) in the calibration, to 0. Some noise still made it through, as can be seen in the predicted projection in Figure 1.

Then, using the gradient prototype with the Adam optimizer with a learning rate of 0.05, a projection was generated. This is shown as the resulting projection in Figure 1. This took 69 iterations in 22.9 seconds. The projection was projected on the black fabric cube, which is also shown in Figure 1.

This experiment did not work with a white cube, since the white cube is already brighter than the mirror cube. Adding more light

[8] https://peach.blender.org
[9] https://www.philips.nl/c-p/SPC1030NC_00/notebookwebcam
[10] https://tweakers.net/pricewatch/305726/infocus-in3118hd/specificaties/

would then only make it less like the mirror cube. This can be seen in Figure 12.

(a) Desired Scene (b) The closest possible result

Fig. 12. A cube with the desired material is on the left. The closest the prototype can get to the desired scene, with a white cube, is projecting nothing. This is because every pixel in the white cube is already brighter then the pixels in the mirror cube.

This prototype converged quite fast. In this experiment, there was no major improvement after ~20 iterations, as can be seen in Figures 13 and 14.

6 DISCUSSION

Several improvements can be made to make the pipeline faster, or more realistic. Some are highlighted in this section.

6.1 Calibration

In a bigger scene, one may want to use a higher resolution projector. But, as can be seen in Table 1, this could take very long, as it is linearly correlated to the number of projector pixels. This could be improved by using different calibration methods. In the future, using Gray code patterns could be considered [10]. The camera pixel to projector pixel mapping can then be acquired by a library like OpenCV's *GrayCodePattern* class[11].

The size of the calibration files is also a problem as can be seen in Table 2. On disk, this can be reduced by compressing, as can be seen in Section 5.2. This is because the matrices mostly contain large strings of zeros. This can be improved by using sparse matrices and tensors in both SciPy and PyTorch. These are more efficient in storing matrices with a large number of zeros[12].

In a real-world scenario, the current way of calibrating can give some artifacts. This is because of noise on the camera, or slightly changing lighting. The model then thinks that a certain projector pixel has an influence on unrelated pixels, which was just noise. For one pixel, this is not relevant, but when adding all this noise together for all pixels, it can break the whole calibration. This was resolved in the real-world calibration by setting every value in the calibration that was below 10 (out of 255) to 0.

6.2 Resolution

The resolution of the projector cannot be very large, since the optimization will quickly take much longer. Especially when the gradient is not provided.

A possible improvement could be optimizing the projection using a lower resolution first and then using that as an initial guess for the next iteration with a higher resolution. This might help with not getting stuck in local minimums, and can be researched in future work.

6.3 Full-colored images

To use this technique for a full-color image, the straightforward solution would be to repeat this technique for red, green, and blue. But this might not give the optimal results, as the Euclidean distance between two RGB colors does not represent the same perceptional color difference. A uniform color space, like CIELAB, would be more suited since the Euclidean distance does represent the perceptional color difference for human eyes [3]. This would yield a more accurate loss and gradient for human perception.

6.4 White cube

In Section 5.6 it was shown that projecting on a white cube did not work. This was because the projector can only add light, and the prototype does not know how to make objects look darker by adding lights. This is possible however since when making parts of the scene brighter, our eyes adjust and other parts appear darker.

This can be solved by having no ambient light in the room, like in a cinema, since then a white cube can appear black.

Another solution is to adjust the camera-projector settings, such that the black of the projector slightly underexposes on the camera, and the white of the projector slightly overexposes on the camera [7]. The optimizer has more freedom to change the scene, and out eyes adjust to that.

Another option is to add a tone-mapping function to the loss function. This converts between the pixel value and the perceived brightness [2].

6.5 Acquiring the desired image

Currently, the desired image was acquired by generating it using the projector and the object, or by having an object with the desired material. Other techniques can be considered as well.

A 3D scan could be made using the same projector-camera setup [10]. Using this scan, a mesh can be built and a material can be applied to that. This can then be used as the desired view.

Something similar to work by Amano [1] can also be done, where they apply image processing to the camera image to generate the desired image. For example, highlights can be enhanced on the image from the camera, and that can then be used as the desired image to make the object look more glossy.

7 RESPONSIBLE RESEARCH

During this research, we tried to adhere to the Communism ethos as much as possible. This entails open communication of methods, results, and knowledge [11]. This can be seen in Section 4 where was explained exactly what kind of libraries, languages, and frameworks were used in the prototypes. In the results section (Section 5), all the parameters and hardware information were stated. This makes results reproducible, transparent, and verifiable. Not only the good but also the more disappointing results were shown. This is to share

[11] https://docs.opencv.org/3.4/dc/da9/tutorial_decode_graycode_pattern.html
[12] https://pytorch.org/docs/stable/sparse.html

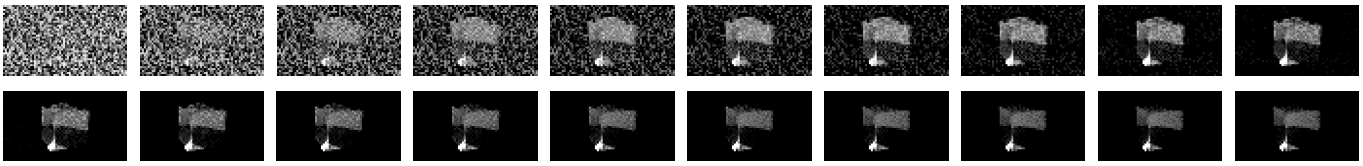


Fig. 13. Every other projection of the first 40 iterations of the minimization. This was for the real-world scene as shown in Figure 1. It converges rather quickly after only 20 iterations.

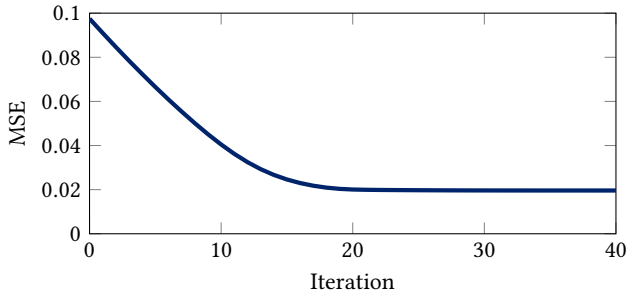


Fig. 14. The Mean Squared Error for the first 40 iterations of the minimization. There is no visible improvement after 20 iterations. This is an average over 10 runs.

the knowledge and make sure other researchers do not have to repeat those same steps.

Universalism partly applies in this research. This means for this project that no personal beliefs and biases can play a role in the evaluation of the research [11]. Some objective measures were given in the results section, like run time, and iterations. But by giving figures with qualitative results for the reader to judge, personal biases are introduced. This could be reduced by doing a user study to what images are actually 'good' or 'bad'.

Disinterestedness partially applies in this research. This is because, in our personal lives, we do not directly gain something if we succeed with these technologies, since we cannot directly apply them in our lives. But, since we get a grade for this project, and a positive outcome could lead to a higher grade, we might be enticed to alter the results.

We tried to have Organized Skepticism during this project. This was achieved by critically looking at shortcomings and limitations. A second pair of eyes was also very useful for this since it is easy to oversee things in your own work.

8 CONCLUSION

In this paper, we have presented a method for changing a scene's appearance by using a projector-camera setup. The prototype uses the camera to measure the influence of every projector pixel in the scene. Using this calibration, the prototype optimizes a projection, so that when it is projected onto the scene, the scene looks as close as possible to the desired scene from the perspective of the camera.

For this, the geometry of the object does not need to be known. This makes it suitable for complex geometries, rather than just plainer surfaces.

However, the calibration can take very long for higher resolution projectors, and the calibration files can grow rather big for higher resolution projectors and cameras. This can be solved in future work by using different calibration methods and storage methods for the calibration matrices.

ACKNOWLEDGMENTS

I would like to thank my responsible professor, Elmar Eisemann, for his ideas, tips, and help with this research project. Next to him, I would like to thank my supervisors, Michael Weinmann and Baran Usta, for their weekly meetings full of tips, tricks, pointers, and ideas.

REFERENCES

[1] T. Amano. 2013. Projection Based Real-Time Material Appearance Manipulation. In *2013 IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 918–923. https://doi.org/10.1109/CVPRW.2013.135

[2] Frederik Anrys, Philip Dutré, and Yves D Willems. 2004. Image-based lighting design. 15.

[3] Jinhui Chao, Ikue Osugi, and Masaki Suzuki. 2004. On definitions and construction of uniform color space. In *Conference on Colour in Graphics, Imaging, and Vision*, Vol. 2004. Society for Imaging Science and Technology, 55–60.

[4] K. Fujii, M. D. Grossberg, and S. K. Nayar. 2005. A projector-camera system with real-time photometric adaptation for dynamic environments. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, Vol. 1. 814–821 vol. 1. https://doi.org/10.1109/CVPR.2005.41

[5] Amir Ghodrati, Efstratios Gavves, and Cees GM Snoek. 2018. Video time: Properties, encoders and evaluation. *arXiv preprint arXiv:1807.06980* (2018).

[6] B. Huang and H. Ling. 2019. CompenNet++: End-to-End Full Projector Compensation. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. 7164–7173. https://doi.org/10.1109/ICCV.2019.00726

[7] B. Huang and H. Ling. 2019. End-To-End Projector Photometric Compensation. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 6803–6812. https://doi.org/10.1109/CVPR.2019.00697

[8] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[9] P. Kurth, V. Lange, M. Stamminger, and F. Bauer. 2020. Real-Time Adaptive Color Correction in Dynamic Projection Mapping. In *2020 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. 174–184. https://doi.org/10.1109/ISMAR50242.2020.00039

[10] Douglas Lanman and Gabriel Taubin. 2009. Build your own 3D scanner: optical triangulation for beginners. , Article 2 pages. https://doi.org/10.1145/1665817.1665819

[11] Robert K Merton. 1973. *The sociology of science: Theoretical and empirical investigations*. University of Chicago press.

[12] M. R. Mine, J. van Baar, A. Grundhofer, D. Rose, and B. Yang. 2012. Projection-Based Augmented Reality in Disney Theme Parks. *Computer* 45, 7 (2012), 32–40. https://doi.org/10.1109/MC.2012.154

[13] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch. (2017).

[14] Ciyou Zhu, Richard H. Byrd, Peihuang Lu, and Jorge Nocedal. 1997. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *ACM Trans. Math. Softw.* 23, 4 (1997), 550–560. https://doi.org/10.1145/279232.279236