Deep Learning for Projector Warp Correction

Shoham Weiss
University of North Texas
shohamweiss@gmail.com

Abstract

In this paper, we present a solution for training deep learning models to compensate for warps in projected images. We create a Unity environment that simulates a projector system and expose it as a python library. This allows models written in python to project images onto a simulated surface and see the resulting warped image. We also use this environment to train a reinforcement learning model to adjust the projected surface to correct for warps. Additionally, we propose a GAN-based supervised learning approach for training models to correct for projected warps. With models trained using this environment, projectors can be used to project onto any surface, enabling a range of augmented reality applications.

Our implementation results are available on GitHub: https://github.com/ShohamWeiss/AIProjectorWarpCorrection.

1. Introduction

Projectors have traditionally required a flat white screen to display images accurately, but recent advances in computer vision and deep learning have made it possible to dynamically adjust the projected image to compensate for warps on non-flat surfaces. In this paper, we propose a deep learning approach for the trained model to adjust the projected image in real-time based on the warped result.

The problem we are addressing is the lack of a dataset for training deep-learning models to compensate for warps in projected images. Additionally, training a model to adjust for warps requires the ability to project the model's output and compare it to the ground truth. To solve these challenges, we developed a Unity environment and python library that allows for the training of any computer vision model written in python as the SDK allows for creating a surface to project on, projecting an image, and extracting the warped projected results from the environment.

Our thesis is that by creating a virtual environment for training deep learning models to adjust for warps in projected images, we can overcome the challenges of collecting real-world data and evaluating the performance of the models. In the following sections, we will describe our approach in detail and present the results of our experiments.

2. Background

Our research was inspired by the work of Bingyao Huang et al. in the work of correcting for warped projected images [1]. In their paper, Huang et al. proposed an end-to-end solution for solving the problems of geometric and photometric disturbance of the projection surface together. They designed a new type of network, called WarpingNet, which uses a cascaded coarse-to-fine structure to learn how to fix the sampling grid of the projected image. They also designed another network, called CompenNeSt, which uses a siamese architecture to understand how the projected image changes when it hits the surface, and to use this information to improve the image. By combining WarpingNet and CompenNeSt, Huang et al. were able to train a single network, called CompenNeSt++, to do both tasks at the same time. In their work, Huang er. al use a physical projector environment to collect their data and run their evaluations of the model. This inspired our work of creating a similar setup but as a virtual environment.

Our approach differs from the work of Huang et al. in several key ways. First, we developed a Unity environment and python library that allows for the creation of a virtual projection environment for training deep learning models to adjust for warps in projected images. This allows us to overcome the challenges of collecting real-world data and evaluating the performance of the models. Second, we trained a reinforcement learning model in the Unity environment to learn to adjust the warped surface so that the projected images end up as flat images. This is a different approach than the one used by Huang et al., which relied on pre-training with synthetic data to reduce the number of training images and training time.

3. Methods

3.1. Building the Virtual Projection Environment

The virtual projection environment used in this study is built using the Unity game engine and a python library that exposes the environment as a python API. This allows for the creation of a virtual wall with a projector projecting an image on it that can be used to train a deep-learning model to adjust for warps in the projected images.

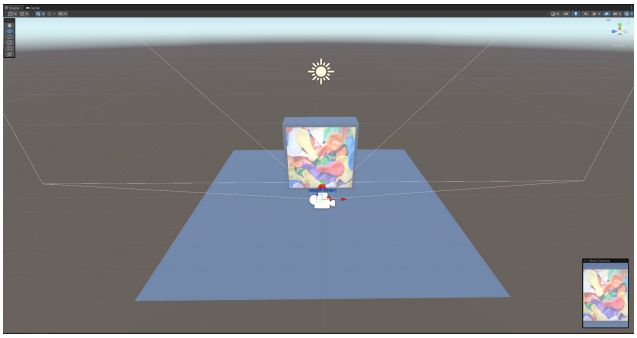

Figure 1. Unity projector environment front view

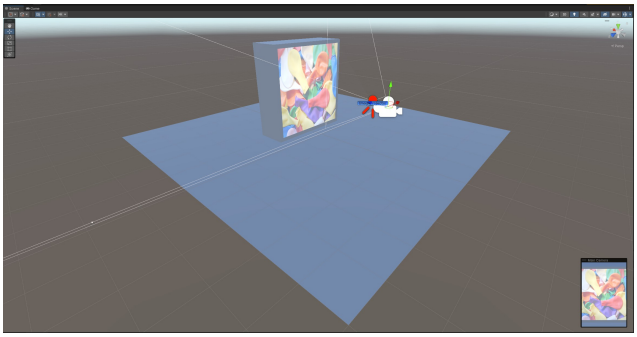

Figure 2. Unity projector environment from the angled view

The Unity environment consists of a wall made up of 9 blocks arranged in a 3x3 grid. The blocks are positioned such that the wall is initially flat facing the camera. A projector and a camera are placed in front of the wall to project images onto the wall's surface and capture images of the projected results.

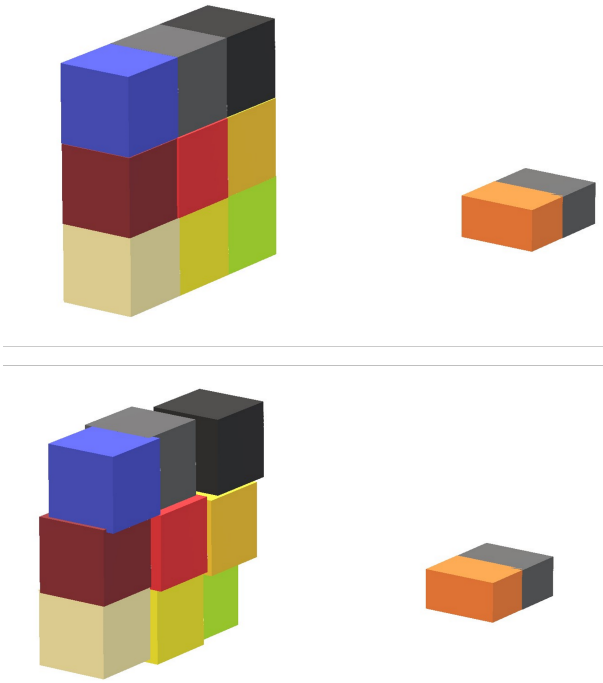

Figure 3. Diagram showing the environment setup. The orange block represents the camera, the grey block represents the projector, and the other colored blocks represent the blocks making up the wall. Top: Environment setup with a flat wall. Bottom: Environment setup with a randomly configured wall.

The user chooses an image to project onto the virtual wall using the python API and can choose to set the wall orientation in the x-axis (axis facing the camera) or set it to randomly orient the wall. Then the user can take a snapshot of the projected result.

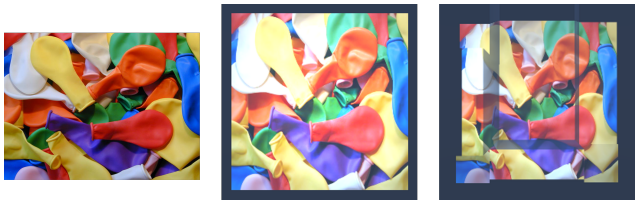

Figure 4. Left: the original image of balloons to be projected. Middle: Projected balloons on a flat wall. Right: Projected balloons on the randomly positioned wall.

The python library provides functions for setting the image to be projected, adjusting the position of the blocks in the wall, and taking a snapshot of the projected image. Here is an example of how the python library can be used to set the image to be projected, set the wall and save a snapshot of the projected result:

```python
# Import the ProjectorEnvironment class from the
# aiprojector library
from ProjectorEnvironment import
  ↪  ProjectorEnvironment

# Create an environment
#     this will use myImage.png as initial image
#     to project
#     it will create a folder with the name of the
#     image and take a snapshot of the flat wall
#     to get a 'label' image
env = ProjectorEnvironment("myImage.png")

# Set a new Image to project
# param new_class:
#     true: new image creates a new folder and
#     label snapshot in that folder (treated as a
#     new class)
#     false: adds the image to the current folder
#     (treated as the same class)
env.NewImage(filepath, new_class=false)

# Create a new random wall to project image onto
env.NewRandomWall()

# Flatten Wall
env.FlattenWall()

# Set wall position to your requirements
wall = {
       "bottomLeft": 0.0,
       "bottomRight": 0.0,
       "bottomMiddle": 0.0,
       "middleLeft": 0.0,
       "middleRight": 0.0,
       "middleMiddle": 0.0,
       "topLeft": 0.0,
       "topRight": 0.0,
       "topMiddle": 0.0
    }
env.SetWall(wall)

# Save a snapshot of current projection on the
# wall
#     saved into the folder of the original image
#     as the folder name
env.Snapshot("snapShotName.png")

# Close the environment to free up memory
env.close()
```

The python library provides a NewImage() function, which can be used to set a new picture to project. The python library provides a newRandomWall() function, which can be used to adjust the position of the blocks in the wall to create a new surface with randomly positioned blocks. Finally, a Snapshot() function, which can be used to take a snapshot of the projected image. These can be run in sequence to create training data. These can also be useful for evaluating the performance of a deep-learning model that has been trained to adjust for warps in the projected image.

3.2. Training the Reinforcement Learning Model

The reinforcement learning model used in this study was trained using Unity's ML-Agents framework. This framework allows for the training of intelligent agents in a simulated environment using reinforcement learning algorithms.

The goal of the reinforcement learning model is to learn to adjust the position of the blocks in the wall in order to flatten the surface and correct for warps in the projected image. At the start of each episode, the blocks are given a random position in the x-axis. The model uses the resulting warped image as input and makes decisions about how to move the blocks in order to flatten the wall.

The specific configuration used for the reinforcement learning model includes the following settings:

```yaml
Trainer type: PPO (Proximal Policy Optimization)
Hyperparameters:
  Batch size: 100
  Buffer size: 200
  Learning rate: 3.0e-4
  Beta: 5.0e-2
  Epsilon: 0.2
  Lambd: 0.95
  Number of epochs: 100
  Learning rate schedule: linear
  Beta schedule: linear
  Epsilon schedule: linear
Network settings:
  Normalization: disabled
  Hidden units: 256
  Number of layers: 4
  Visual encoding type: simple
Reward signals:
  Extrinsic:
    Gamma: 0.99
    Strength: 1.0
Maximum number of steps per episode: 300,000
Time horizon: 64
Summary frequency: 1,000
```

In this paper, we propose a training method for deep learning models to correct for warps in projected images. Our method uses a reinforcement learning approach, in which the model is exposed to a series of episodes in which it must adjust the position of blocks to flatten a simulated wall and produce a corrected, unwarped projected image. We evaluate the model's performance based on its ability to correctly flatten the wall and produce a corrected image.

We focus on two sub-scenarios for our training: placing one block to flatten the wall, and placing two blocks to flatten the wall. In the first scenario, we keep eight of the nine blocks flat and randomly vary the position of one block. The model is then given 300 steps (5 seconds) to place the block so that it aligns with the rest of the wall, creating a flat image. In the second scenario, we do the same but keep seven blocks flat and move two.

The model takes in an image of size 84x84 as input. For the front layers, we use a pre-trained CNN provided by Unity's mlagents library. The middle layers consist of four fully-connected layers of size 256, and the final output layer is size 3 for the one-block scenario. This output layer represents the movement of the block, with each value representing the movement in the forward, backward, or no movement direction. This allows the model to learn to adjust the position of the block to correct for warps in the projected image. For the two-block scenario, the output layer is size 6, with three movement options for each block.

We structured the reward to be the sum of the distance of the blocks from their "flat position", the position all blocks start at to make the image flat.

$$Reward = -\sum_{i=0}^{n} |blockposition_i - flatposition|$$

Overall, the use of Unity's ML-Agents framework and the specified configuration for the reinforcement learning model enables the training of the model in the simulated environment. This allows for the collection of data and evaluation of the model's performance in a controlled, virtual setting.

3.3. Pix2Pix approach

The reinforcement learning approach to training a model to correct for warps in projected images is not scalable because it is limited by the number of moves that can be made in the environment. In the current implementation, the model can only move a limited number of blocks to flatten the wall, and adding more blocks would require exponentially more training data and computational resources. This makes it difficult to apply the reinforcement learning approach to more complex scenarios with more blocks or other factors that affect the projected image.

On the other hand, a pix2pix-based approach is more scalable and adaptable, as it does not rely on a fixed number of moves or a pre-defined network architecture. It can learn from paired images and generate output images that are similar to the target image, allowing it to adapt to different environments and settings. This makes it a more promising approach for training a model to correct for warps in projected images.

A potential method for training a pix2pix model to correct for warps in projected images is as follows:

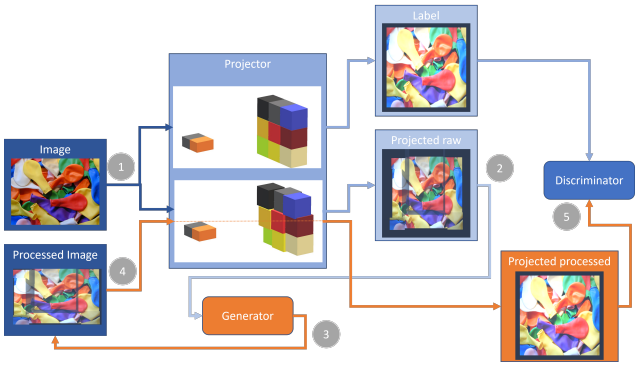

Figure 5. Steps to train a pix2pix model to correct for warped projected images.

1. Project image on flat wall capture projected result as label.
2. Set the orientation of the wall randomly and capture projected result.
3. Pass the projected image through the generator to produce processed image.
4. Run the generator's output through the projector and save the resulting processed projected image.
5. Provide the discriminator with the label image and the processed projected image.
6. Simultaneously train the discriminator to be able to distinguish between the flat and non-flat images and the generator to produce more convicing warp corrections.

This approach leverages the pix2pix model's ability to learn from paired images and generate output images that are similar to the target image. By providing the model with a ground truth flat image and a projected image as input, the generator can learn to produce output images that, once projected, result in a flattened image. The discriminator can then be trained to distinguish between the ground truth and generated images, allowing the generator to improve its output over time.

4. Results

For our results we focus on the reinforcement learning model. There are two ways to evaluate the reinforcement learning model's performance, through metrics and visually. The metrics to evaluate the model are the cumulative reward, the value loss, and the policy loss collected as the model is training. We expect the cumulative reward to be increasing, the polity loss to be decreasing, and the value loss to increase and then decreasing during successful training sessions. Visually we expect to see the blocks moving

initially randomly, and over time seeing the agent become more intentional with its placement of the blocks landing them close to the rest of the blocks.

As seen in figure 6, both the 1 block and the 2 block system seem to have improved the cumulative reward in the first 15 thousand steps, then they decreased the cumulative loss back down. This suggests that the models did not improve in a significant way in the 50 thousand steps given.

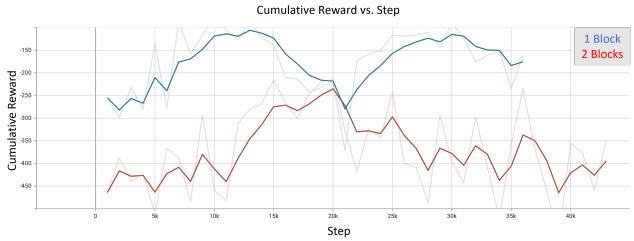

Figure 6. Cumulative reward of the two models during training.

As seen in figures 7 and 8, although the cumulative sum of the reward did not improve over the 50 thousand steps, the variance in the reward did decrease and concentrated more towards the high reward results. This suggests that the models were learning to behave in a more coherent way following better cumulative reward behaviors.

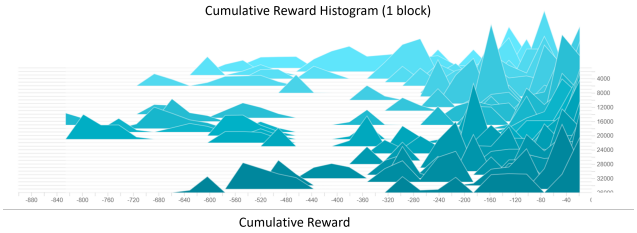

Figure 7. Cumulative reward histogram for the 1 block training showing how often we get which reward during training (z-axis).

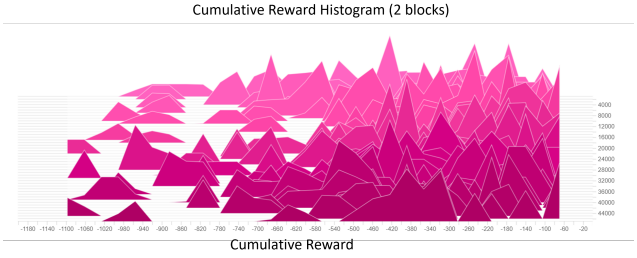

Figure 8. Cumulative reward histogram for the 2 blocks training showing how often we get which reward during training (z-axis)..

In figures 9 and 10, we see the same trend as in figure 6. The models were decreasing the value loss in the first 15 thousand steps and then going back to increasing the loss. Since for the policy loss we expect a slight increase then slight decrease, the continuing increase follows the same trend. The models appeared to have enough time to learn some behavior to follow but not enough time to fully increase their reward to their full potential.

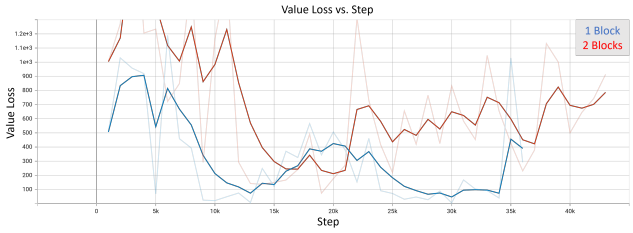

Figure 9. The value loss of the 1 block and 2 blocks models during training.

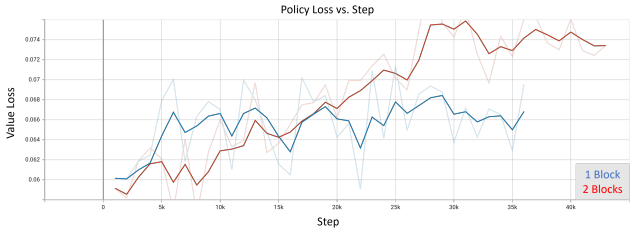

Figure 10. The policy loss of the 1 block and 2 blocks models during training.

These metrics follow a seen behavior in the simulation of the models. In the early episodes, the blocks move sporadically, a lot of time straight back or forward and away from the rest of the wall. This behavior results in really low cumulative rewards. The model needs to first learn to move the block slightly to keep it close to the wall. This is exactly what is seen in action, by the later episodes, and even by step 15 thousand, the model no longer moves the block very far away from the wall and keeps the block somewhat close to the wanted position. Still the model does not always successfully align the block flat with the wall by step 50 thousand.

5. Discussion

In the discussion of our results, we focus on the performance of the reinforcement learning models. We observed that the models were able to learn to keep the blocks close to the wall, but did not have enough training steps to learn how to place the blocks perfectly flush with the wall. We believe that increasing the number of training steps would allow the models to learn this behavior. Additionally, we propose modifying the reward function to use a negative

quadratic offset, which would incentivize the model to place the blocks closer to the center (flush with the wall) for higher rewards. The reward function could be expressed as:

$$Reward = \sum_{i=0}^{n} -(blockposition_i - flatposition)^2 + offset$$

where (blockposition - flatposition) is the distance of the block from the center (flush with the wall) and offset is a constant value that determines the range of the reward function.

Once the model is able to create semi-flush wall orientations for two blocks, we propose increasing the number of blocks until all nine blocks are oriented semi-well by the model. We believe that training a discriminator on the resulting projections could then be used to further train the reinforcement learning model. The discriminator can be used to create a new reward function based on its ability to distinguish between the ground truth and generated images. This new reward function can be used to train the reinforcement learning model to produce even better results, potentially achieving a perfectly flush wall.

Overall, we believe that focusing on developing a pix2pix model using the python API of the virtual projector environment is a promising path forward. The pix2pix model has the potential to be more scalable and adaptable than the reinforcement learning approach, and we believe it could be used to train a model to correct for warps in projected images effectively.

6. Conclusion

We successfully created a virtual projector environment in the Unity game engine and trained reinforcement learning models on said environment. Furthermore, we exposed the virtual environment as a python API that anyone can make use of for training and evaluating other projector warp correction models. We believe that a pix2pix approach would be ideal to go further with this problem and believe that once solved this challenge can have a big impact on the field of augmented reality.

References

[1] Huang, B., Sun, T., Ling, H. (2021, January 7). End-to-end full projector compensation. arXiv.org. Retrieved December 13, 2022, from https://arxiv.org/abs/2008.00965v3 1

[2] Unity-Technologies. (n.d.). Unity-Technologies/ML-Agents: The Unity Machine Learning Agents Toolkit (ML-agents) is an open-source project that enables games and simulations to serve as environments for training intelligent agents using deep reinforcement learning and imitation learning. GitHub. Retrieved December 13, 2022, from https://github.com/Unity-Technologies/ml-agents

[3] Pix2pix: Image-to-image translation with a conditional Gan : Tensorflow Core. TensorFlow. (n.d.). Retrieved February 26, 2022, from https://www.tensorflow.org/tutorials/generative/pix2pix