Augmented Perspective

Jack He, Ryan Kirk, Eric Lou
Stanford University
siyuanhe, rkirk10, erlou @stanford.edu

Abstract

Recent developments in monocular depth estimation have shown promising results in accurately inferring scene geometry from single 2D images. In this project, we investigate and evaluate the limitations of this technology through two processes. 1: The generation of alternative 2D images viewed from different perspectives when given a single 2D image. 2: A comparison of the 'augmented perspectives' produced by two popular monocular depth estimation algorithms. While qualitative evaluations show the value and accuracy of using monocular depth estimates in creating augmented perspectives, the limitations are also clear.

1. Introduction

Depth estimation is a powerful technology with a myriad of applications, but typically requires multiple photos of the same scene from different angles to create an accurate representation. Thus, there have been many recent advancements that investigate the accuracy and applicability of monocular depth estimation. Our research focuses on monocular computer vision.

We first want to test whether using a depth map in conjunction with computer vision geometric principles is sufficient in generating a new 2D image from a different viewing angle, which we call an *"Augmented Perspective"* task. More concretely, given a single 2D RGB image of a scene, we want to generate an image of the same scene when viewed from a different perspective as characterized by some transformation matrix T. Of course, such a generated image would include empty pixels where there were occluded objects. Assuming a known camera projection matrix, we map the pixels in the 2D image into 3D space with a calculated depth map, apply the transformation matrix T, and reproject back into 2D to create a new RGB image.

Second, we want to leverage a set of two popular monocular depth estimation algorithms to compare their augmented perspective accuracy side-by-side. Specifically, Monodepth 2[8] and Content-Adaptive Multi-Resolution Merging (a.k.a. Boosting)[10].

The combination of these two questions allows us to test the current boundaries and limitations of monocular depth estimation algorithms and determine which, if any, is the best fit for related tasks like image parallax.

2. Related Research

Here, we first review some of the popular monocular depth estimation algorithms, and then review other attempts at the 'augmented perspective' task.

2.1. Monocular Depth Estimation Algorithms

2.1.1 Monodepth2

This leading algorithm comes from UCL, Caltech, and Niantic: Monodepth2. They take a self-supervised approach that seeks to minimize the photometric reprojection error. Data collection comes in the form of stereo training. Prior algorithms have not performed well when estimating depth maps with objects in motion. Monodepth2 introduces a feature that automatically detects all of the pixels in the image that contains a moving object and includes this image mask in the training to significantly improve depth estimation, as an improvement from Monodepth[7], its earlier version.

2.1.2 Multi-resolution Merging (Boosting)

Recently, researchers from Simon Fraser University and Adobe Research suggested that besides full image loss from the actual depth map, resolution of the depth estimation algorithm also matters. The proposed double estimation method optimizes for both full image estimation accuracy and patch section estimation accuracy. We call it "Boosting" as it boosts depth map to higher resolution. We hope that higher resolution depth map could generate more accurate point clouds that lead to better reprojection results.

2.1.3 Other Monocular Depth Estimation Algorithms

Common image processing libraries also have started including depth estimation algorithms. Popular library Keras[3] open sourced a depth estimation algorithm in

2021. The ease of use and limited barrier to entry for this library speaks to the prevalence of monocular depth estimation algorithms.

2.2. Augmented Perspective Attempts

2.2.1 Neural Radiance Fields

NeRF[11] is a recent effort trying to extract 3D scene from 2D images. The algorithm calculates camera rays from viewed pixels and and generate volumetric data that can be used to project 3D outputs. In the project demo, the authors are able to generate realistic 3D videos from 2D images. However, this project has to use more than one 2D images in order to generate 3D scenes. In our project, we are trying to generate 3D scenes with only a single image.

2.2.2 SynSin

SynSin[12] is another effrot trying to create short videos according to a user defined trajectory from a single still image. This paper uses a depth map to generate point cloud, reprojects point cloud to different 2D views according to transformation matrix, and uses a refinement network to fill in the occluded pixels. While very similar to our approach, this project uses its own depth map estimator and their results only show images viewed from slightly different angles. In our project, we are trying to reuse existing depth estimation models and generate augmented images viewed from larger angle differences with the caveat of occluded pixels. Being able to swap depth models helps our project to improve without retraining.

2.2.3 Other augmented view generation work

Besides NeRF and SysSin, Li *et al*. [9] tried to generate images with different focuses from a single image; Evain *et al*. [4] and Flynn[5] tried to generate stereo images from a single image. Overall, multi-view generation from single images seem to be a well studied problem and many of them requires depth map generation as a prior condition. Hence our project focuses on comparing how augmented perspective generation is affected by the choice of depth estimation model and what we can do without using an occlusion filler.

3. Approach

3.1. Dataset

We use images from the KITTI[6] dataset. Our ideal dataset would have various 2D images of the same 3D scene taken from different angles where the transformation matrix T between captures is published. While Neural Radiance Fields (NeRF)[11] has a dataset that perfectly encompasses this, they do not publish the camera intrinsics used, which is essential for the initial 2D to 3D mapping in an augmented perspective task. While there are deep learning based single image calibration models like DeepCalib[1], it adds unnecessary complexity to this project and would be hard to handle within the time-frame of this project. Moreover, most of the NeRF images were computer generated and had transparent background colors. This poses as a second challenge in adopting NeRF as our dataset as monocular depth algorithms expect a full, realistic scene. Because of this, we ultimately chose KITTI as the best candidate.

Figure 1: Sample image from KITTI dataset

3.2. Evaluation

Given the challenges of empty pixels in the generated 2D image and scaling and lighting differences between the three algorithms and the ground truth image, we will use the following evaluation scheme. Instead of comparing pixel-by-pixel differences, we will run each output image through an edge detection model to pull only the most essential frameworks out of the images. Moreover, since we do not have ground truth images after the reprojection, we plan to reproject a augmented image back to zero angle and compare with input image to see whether a rotated image can be restored to the original. Lastly, the augmented images will be evaluated qualitatively as well.

4. Technical Approach

4.1. Compute Depth Map from Input Image

From a software perspective, we create a `DepthModel` class that serves as a generic entry point to interact with each of the three algorithms. Each algorithm has its own `DepthModel` subclass that bridges the gap to the algorithm-specific implementations. Each image is evaluated by each depth model and the output is saved to a common output directory with model-specific file names.

As expected, monocular depth estimation algorithms cannot predict the absolute depth of every pixel in an image in length units like meters or inches. Instead, they predict it up to a scale factor. As shown above, Monodepth 2 and Boosting use different scales to represent the final depth. Monodepth 2 outputs inverse disparity values from the last Sigmoid step in its network, which is proportional to depth. Boosting, on the other hand, outputs an inverse scaled depth from 1.0 to 0.

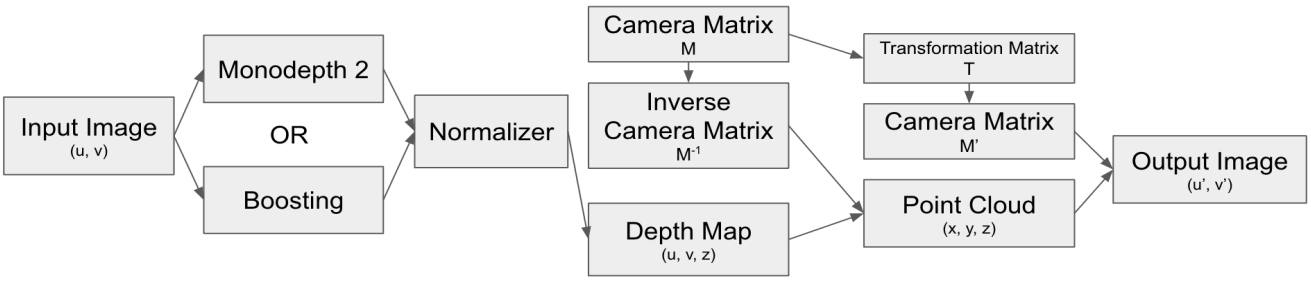

Figure 2: Our processing pipeline.

As result, the colorized output from each model differ by a lot, as shown in Figure 3. In the output of Monodepth 2, closer pixels are represented with darker colors; whereas in Boosting, closer pixels are represented with brighter pixels due to inverse depth. This poses a challenge to our subsequent point cloud generation algorithms as it expect single channel depth map with closer pixels represented by darker colors (i.e. smaller value in greyscale).

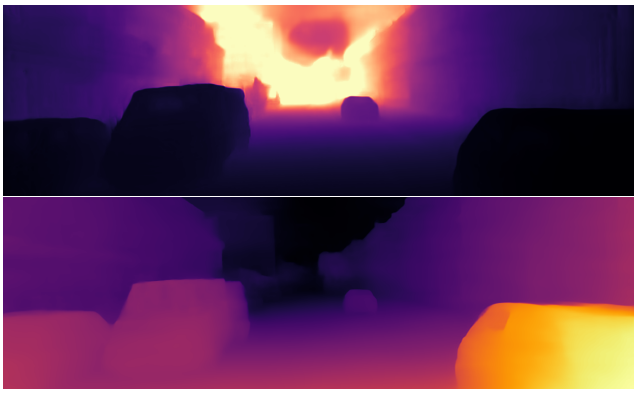

Figure 3: Colorized depth maps from Monodepth 2 (top) and Boosting (bottom)

4.2. Normalizer

To resolve both issues, we made attempts to normalize the two depth maps. Firstly we scale the inverse disparity output of Monodepth 2 from 1 to 1000. 0 is avoided to prevent divide-by-zero problem in later steps. Then we also scale the output of Boosting model from 999 to 1 and deduct it from 1000 to have an output range of 1 to 1000 as well. Lastly, although both depth maps are scaled from 1 to 1000, their distribution is still very different.

As shown by Figure 4, depth from Monodepth 2 is mainly squeezed around 0 to 20; whereas depth from Boosting is more evenly distributed from 1 to 1000. This creates problem in later steps when applying transformation matrices as different depth distribution requires different translational values. Therefore, we normalized both depth distribution so that their 95 percentile are the same. This is effectively dividing depth output from Boosting by 54.8. As shown in Figure 5, output from both models are normalized to the same range. Figure 6 represents final single channel greyscale depth map from both models with closer pixels represented by darker colors.

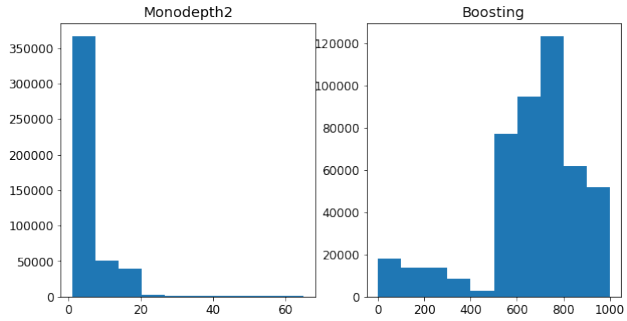

Figure 4: Histogram of scaled depth from Monodepth 2 (left) (95th percentile = 17.6 and Boosting (right) 95th percentile = 963.9

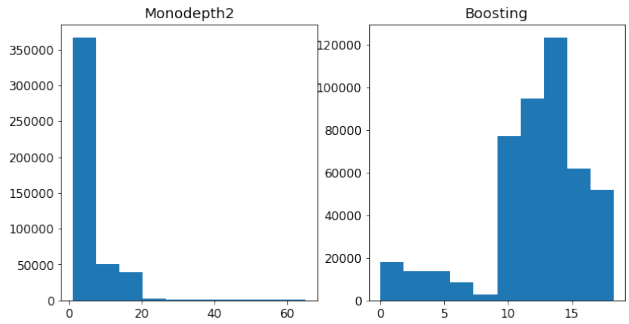

Figure 5: Histogram of normalized depth from Monodepth 2 (left) and Boosting (right) with the same 95th percentile

4.3. Generating Point Cloud

With depth map formatted in (u, v, z) and a known camera matrix M, we were able to calculate a 3D point cloud in

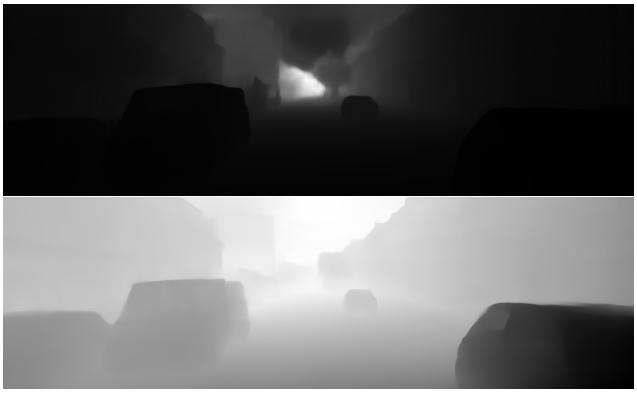

Figure 6: Normalized greyscale depth maps from Monodepth 2 (top) and Boosting (bottom)

(x, y, z) format using Equation 1.

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = zM^{-1} \begin{bmatrix} u \\ v \\ 1 \\ 1/z \end{bmatrix} \tag{1}$$

4.4. Construct Transformation Matrix

The next step would be transforming the camera matrix such that the same 3D world scene can be reprojected to a different perspective. In the transformation matrix, we had to consider both rotations and translations since rotations alone would cause image to go out of scope. Instead we should translate and rotate the image through the following steps to make sure the scene stays mostly in the middle.

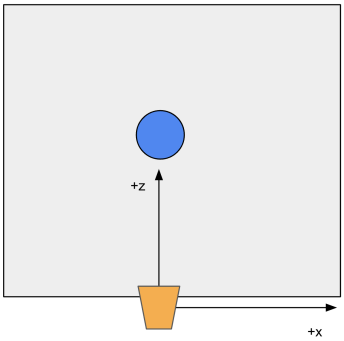

Figure 7: Initial Camera Position

From initial camera position in Figure 7, we first translate the image to the right as shown in Figure 8, which is negative x-translation. This gives room for the camera to rotate while keeping the object in the center.

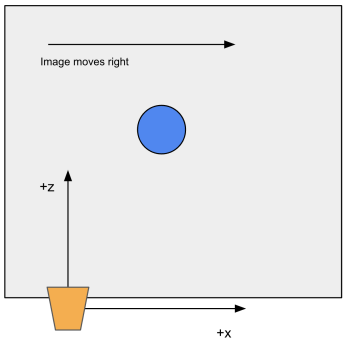

Figure 8: Move Horizontally

From Figure 8, we then rotate the camera, in this example, about +y axis by 15 degrees. As shown in Figure 9, the object remains in the center of camera.

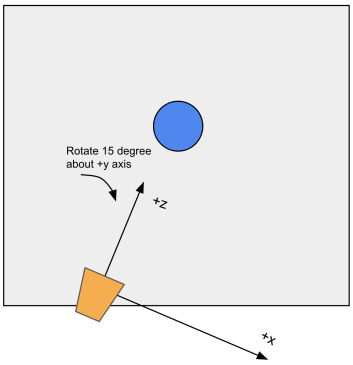

Figure 9: Rotate Camera

After applying translations in addition to rotations, we obtain the actual transformation matrix like the example in Equation 2 with 15 degree +y rotation and -0.3 x translation.

$$T = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.966 & 0 & 0.259 & -0.3 \\ 0 & 1 & 0 & 0 \\ -0.259 & 0 & 0.966 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

4.5. Reproject Image

Then we obtain the updated camera projection matrix $M' = MT$ and then apply M' to all 3D coordinates, P_n, from the point cloud to obtain the reprojected image coordinates $p'_n = (u'_n, v'_n)$. Then we make the assignment that pixel color at $p'_n = (u'_n, v'_n)$ is the same as the pixel color at $p_n = (u_n, v_n)$ in a newly created numpy array with the same dimension as the original input image for all three RGB channels. When reprojected pixels falls outside the image frame, they are ignored. Moreover, it is possible for multiple 3D points to be reprojected to the same 2D image

coordinates. In this case, the closer 3D point always wins. Please see our Python code below for the detailed algorithm.

```
M_square = np.diag(4)
M_square[:3, :] = M
M_inv = np.linalg.inv(M_square)
M_new = M.dot(RT)
H, W, C = image.shape
new_image = np.zeros_like(image)
depth_color = {}
for u in range(W):
  for v in range(H):
    z = depth_map[v, u]
    img_h = np.array([u, v, 1, 1.0 / z])
    world_p_h = z * M_inv.dot(img_h.T)
    world_p_h /= world_p_h[-1]
    p_ = M_new.dot(world_p_h)
    p_ /= p_[-1]
    u_ = round(p_[0])
    v_ = round(p_[1])
    if u_ < 0 or u_ >= W \
        or v_ < 0 or v_ >= H:
      continue
    if (v_, u_) not in depth_color \
        or z < depth_color[(v_, u_)]["depth"]:
      depth_color[(v_, u_)] = \
          {"depth": z, "color": image[v,u]}
for k, v in depth_color.items():
  new_image[k[0], k[1]] = v["color"]
```

4.6. Image-level Post-processing

Figure 10: Original reprojected image (top) and final filled image (bottom)

By the nature of our reprojection pipeline, adjacent pixels in the original image may reproject to the same pixel in the augmented image. This manifests in random, empty black pixels in the new image. So, we counteract this effect while making sure to maintain the integrity of the parts of the image that should indeed contain empty pixels. After the image is reprojected, we fill all of the empty pixels that have at least 4 (out of 8 total) colored neighboring pixels with their average color. This ensures that empty pixels due to occlusions stay empty while as many unexpected empty pixels are filled. This filled image is the final product shown in the following section.

5. Experiment

5.1. Experimental Setup

We ran each of five images from the KITTI dataset through our augmented perspective pipeline to predict what the image would look like if the camera were panned up to 25 degrees and translated to keep the scene centered. More specifically, we imagine a camera in the reconstructed 3D scene smoothly panning from the left side of its original position (while keeping the scene centered), to its original position when the original photo was taken, to the right side of its original position (while keeping the scene centered).

The estimated image on the extreme left hand side was a 25 degree clockwise rotation and a 0.6 unit translation leftward. The estimated image in the center had no rotation or translation. The estimated image on the extreme right hand side was a 25 degree counter-clockwise rotation and a 0.6 unit translation rightward.

In this entire range, 60 frames were estimated and stitched together to create a GIF. In the end, for each of the five images, we had one GIF that utilized Monodepth 2 and another GIF that utilized Multi-resolution Merging.

5.2. Qualitative Results

We first discuss results for Monodepth 2 and Boosting separately, and then consider them in comparison with each other. Visit our public Google Drive folder[1] for a copy of all of our final reprojected images.

5.2.1 Qualitative Results for Monodepth2

Below, we show a sample of 5 equally-spaced images from the entire hue of the 60 total frames for Monodepth2. For a full view of the GIF, visit a sample on GitHub[2].

These images are very promising and make sense. Starting from the original image in the center, as the camera rotates clockwise and pans left, we move towards the image on the top. The car in the bottom right pulls away from the scene because it is at a closer depth of field than the buildings in the background. It makes sense for those buildings to retreat out of view faster. We also notice that by the end of the 25 degree rotation and the translation, the orange vehicle is now aligned parallel with the camera's field of view. The side of the vehicle is no longer visible, which also makes sense. There are indeed some noisy artifacts that are a byproduct of our vision pipeline. For example, in

[1] https://tinyurl.com/augmented-perspective

[2] https://github.com/hsysuper/augmented-perspective/blob/main/kitti3_output.gif

Figure 11: Sample of predicted augmented perspective images under Monodepth2 (clockwise rotations from top to bottom in degrees: 25, 12.5, 0, -12.5, -25)

the orange vehicle's windshield, there are gray/white pixels that bleed through the expected black pixels. After reprojecting the pixels in the 3D world back into 2D to create this new image, there may not be a black pixel from the original windshield that projects into the new image. In fact, a gray pixel from the building in the background may end up being the only pixel that projects into the windshield's spot.

Starting from the original image in the center, as the camera rotates counter-clockwise and pans right, we move towards the image on the bottom. We see a similar effect showing up on the left side where the vehicles begin to protrude. We also notice that the camera is now panning over the hood of the car in the bottom right, much like a cinematic shot would entail.

Figure 12: Sample of predicted augmented perspective images under Boosting (clockwise rotations from top to bottom in degrees: -12.5, -25)

5.2.2 Qualitative Results for Boosting

Below, we show two images for Boosting that were created under the same conditions as the last two images from Figure 11.

Here, we see that the car in the bottom left protrudes out of the new image as it is at a closer depth of field than the store in the background. We also notice that the car in the bottom right is no longer present after a 25 degree rotation and small translation, implying that it is no longer in sight. While possible, we expect at least a part of the side or hood of the car to remain visible, simply by estimation.

5.2.3 Overall Qualitative Results

It is evident that Monodepth 2, on average, placed objects at a closer depth than Boosting did. The cars protruded less in the latter algorithm, and there were also very few empty black pixels in the image. We also do not see the noisy artifacts that we saw in Monodepth 2 where the gray/white pixels bled through the orange vehicle's windshield. This is also attributed to the fact that the vehicle is predicted to be at a large depth. The greater the depth of the pixels on the vehicle, the more likely they'll reproject to the expected location in the new image.

5.3. Quantitative Results

5.3.1 L2 and SSIM Distance to Augmented Original

Due to the lack of a extensive image dataset, with included camera intrinsics and without the use of transparent pixels, we faced difficulties when generating meaningful quantitative results. As a workaround, we used a single image with known camera intrinsics from KITTI and applied our algorithm to generate a augmented perspective of the original image. We then applied our algorithm to the augmented perspective to generated an estimated perspective from the

same point of view as the original image. Then using the original image as our ground-truth, we computed the L2 difference between our `augmented_original` perspective and the original image.

From our results we find that the images produced by the boosting algorithm are on average 18.38 % closer to the ground-truth original image (L2 w/ edge detection).

After generating two augmented perspectives, our images had a significant number of occluded pixels. To mitigate the differences caused by the occluded pixels, we applied Canny edge detection[2] to focus on the position of objects in our image. After doing so, the Boosting algorithm produced images closer to the reference (original) image. This contradicts our qualitative results as the Monodepth 2 algorithm produced more realistic results. An explanation as to why the Boosting algorithm produced images that are closer to the reference could be due to Boosting placing objects further away in the image. As a result, each object has few occluded pixels and the overall image has significantly fewer occluded pixels and is closer to the original image.

	L2	L2(Edge)
kitti1.png	10.2×10^7	9.6×10^4
kitti2.png	9.6×10^7	9.2×10^4
kitti3.png	8.8×10^7	6.7×10^4
kitti4.png	9.0×10^7	10.6×10^4
kitti5.png	11.2×10^7	16.0×10^4

Table 1: L2 Diff Monodepth2 (Raw RGB vs Edge Detection)

	L2	L2(Edge)
kitti1.png	7.8×10^7	6.1×10^4
kitti2.png	9.5×10^7	8.3×10^4
kitti3.png	8.4×10^7	6.3×10^4
kitti4.png	7.8×10^7	7.8×10^4
kitti5.png	9.7×10^7	14.6×10^4

Table 2: L2 Diff Boosting (Raw RGB vs Edge Detection)

	SSIM(raw RGB)	SSIM(Edge)
kitti1.png	0.5067	0.6758
kitti2.png	0.4402	0.7053
kitti3.png	0.5298	0.7466
kitti4.png	0.4366	0.6095
kitti5.png	0.4603	0.5773

Table 3: SSIM Between Monodepth 2 and Boosting (Raw RGB vs Edge Detection)

t

There is also a significant disparity in runtimes between the boosting and Monodepth2 algorithms. For the Monodepth, it takes about two seconds to generate a depthmap of the scene while boosting takes about 112 seconds to generate a depthmap of the scene.

5.3.2 Timing

While Boosting produces higher resolution results, it takes about 112 seconds on average to generate a depth map for the KITTI dataset using a Google Colab Pro High RAM GPU instance in March 2022. In comparison, it only took Monodepth 2 around 2 seconds on average to generate a depth map for an input image. This makes Boosting much less appealing to real time or near real time use cases. Lastly, augmenting each image took about 11 seconds on the same compute platform. However, this can be optimized faster by vectorizing our algorithms and offloading part of our algorithm to the GPU.

6. Conclusion

An overview of our project shows promising results, that given a single image we are able to generate a realistic augmented perspective, aside from the inclusion of occluded pixels. Given the short time frame of the project, there are additional improvements to our project we would have liked to make. As the Boosting algorithm produces high resolution depth maps, we would have like to improve the normalization of the depth map to yield better augmented perspectives. In addition, the 11 seconds it takes to generate an augmented perspective can be improved with the use of vectorized coding techniques and methods. These improvements can significantly improve the quality of our outputed images.

Our code is available on GitHub[3] and is available to view upon request.

References

[1] O. Bogdan, V. Eckstein, F. Rameau, and J.-C. Bazin. Deepcalib: a deep learning approach for automatic intrinsic calibration of wide field-of-view cameras. In *Proceedings of the 15th ACM SIGGRAPH European Conference on Visual Media Production*, 2018.

[2] J. Canny. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, (6):679–698, 1986.

[3] F. Chollet et al. Keras. `https://keras.io`, 2015.

[4] S. Evain and C. Guillemot. A lightweight neural network for monocular view generation with occlusion handling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(6):1832–1844, 2019.

[5] J. Flynn, I. Neulander, J. Philbin, and N. Snavely. Deepstereo: Learning to predict new views from the world's imagery. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5515–5524, 2016.

[3] https://github.com/hsysuper/augmented-perspective

[6] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[7] C. Godard, O. Mac Aodha, and G. J. Brostow. Unsupervised monocular depth estimation with left-right consistency. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 270–279, 2017.

[8] C. Godard, O. Mac Aodha, M. Firman, and G. J. Brostow. Digging into self-supervised monocular depth estimation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3828–3838, 2019.

[9] Q. Li and N. Khademi Kalantari. Synthesizing light field from a single image with variable mpi and two network fusion. *ACM Transactions on Graphics*, 39(6), 12 2020.

[10] S. M. H. Miangoleh, S. Dille, L. Mai, S. Paris, and Y. Aksoy. Boosting monocular depth estimation models to high-resolution via content-adaptive multi-resolution merging. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9685–9694, 2021.

[11] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*, pages 405–421. Springer, 2020.

[12] O. Wiles, G. Gkioxari, R. Szeliski, and J. Johnson. Synsin: End-to-end view synthesis from a single image, 2019.