An Image Augmentation Pipeline With Customized Data For AR Education

Haochen Zhang
Harvard University
Cambridge, Massachusetts, USA
haochen_zhang@gsd.harvard.edu

Fangqing He
Harvard University
Cambridge, Massachusetts, USA
quinnhe@gsd.harvard.edu

Abstract

As with the introduction of time of flight (ToF) camera to modern mobile devices, we saw great prosperity in the AR field, especially the well-developed AR APIs for developers such as ARKit, ARCore. However, most of them are kept as black boxes. The learning curve of the AR field, which often involves mathematics, optical physics, geometry, and computer software technology, can be too high for beginners. Thus, a system that allows learners to use their own customized data, step-by-step explicit implementation, and eventually an ability to explain is needed.

1. Introduction

AR, known as the Augmented Reality (AR) technique, builds a connection between the virtual universe and reality. The AR technique facilities the development in various areas including Pedagogy[10, 6], Medicine[4], Entertainment[9], Fashion[11] etc. However, the wide usage of AR does not demystify the technology for the public. Even for some experienced developers, the underline mechanism of AR remains unclear, and they depend on well-developed AR development tool kits such as ARKit, and ARCore to develop projects. For learners who are interested in understanding how AR works, they have to undergo the high-math-demanding theory study, which inevitably raises the bar of learning AR. A learning tool that allows AR learners to verify the learned theory with practice, and experiment with their own customized data would greatly lower the learning barriers and demystify AR for a wider audience.

The project aims to build a pipeline that allows learners to build their very first AR project from scratch with customized data. The pipeline covered the basic steps of a simple AR project, including camera calibration, camera pose estimation, ray tracing, and lighting estimation. Furthermore, the pipeline explored two approaches to improve the efficiency of the augmentation 1) High Resolution Speculation 2) Bounding Box Trim, intending to provide learners some inspirations on AR system optimization.

1.1. Ray Tracing

Ray tracing is the footstone of modern computational rendering. It is a technique for modeling light transport for use in a wide variety of rendering algorithms for generating digital images. However, ray tracing is a process that includes heavy computation. Especially with the growing resolution of the cameras and screens (i.e., the resolution of iPhone 13's screen is 2532 * 1170), optimization becomes increasingly essential for ray tracing implementation [5]. The pipeline aims to reveal the disadvantage of ray tracing to learners through experiments in high resolution speculation and bounding box trim.

1.2. Lighting Estimation

A more generalized method to estimate light with high fidelity is based on Monte Carlo ray tracing, which involves more depths of reflection. However, the bulky calculation limits its application in AR, like real-time rendering. To simplify, one could use a presumptive bidirectional reflectance distribution function (BRDF) model, where the geometry surface is assumed to be Lambertian, and the light is directional [8]. Under the particular condition, reflectance can be reduced to the multiplication of surface normal vector and incident light vector.

Another possible calculation-consuming problem comes from the type of model surface. To some extent, triangle mesh can significantly reduce the ray-tracing calculation; however, its considerable memory cost limits its usage in mobile devices. As for the non-uniform rational basis spline surface (NURBS), which is much lighter, the trimming algorithm is still cumbersome [7]. Here, we would like to explore a more simplified way to reduce the unnecessary calculation, rasterization, which tries to treat the newly calculated reflectance values as an original picture, and remove "noise" among them (Non-local means denoise).

2. Methods

We created the image augmentation pipeline based on a python notebook AprilBoardCalibration.ipynb provided by cs283 instructor team [2]. The notebook offers functions to calibrate the camera and estimate relative pose utilizing April Boards, planar calibration boards created by the April Robotics Laboratory [1].

Our image augmentation pipeline included: setting up scenes and capturing customized data with smartphones, recovering camera matrix and finding dominant 3D ground plane with April Boards, and rendering virtual objects with diffuse lighting. The pipeline was evaluated with both customized images captured ourselves and images provided by the cs283 instructor team [2].

2.1. Camera Calibration

Our pipeline reused the implementation in AprilBoardCalibration.ipynb for camera calibration. To illustrate in detail, we would explain the process step by step. The pipeline first set up April tag detector provided by pupil_apriltags library with default parameters. Then, for each image, the pipeline detected the April tags and got `imgpoints` the 2D image locations of tag centers and `objpoints` their corresponding 3D coordinates in the broad's coordinates. Lastly, the pipeline utilized `cv2.calibrateCamera` from OpenCV to calibrate the camera, and output `calMatrix` calibration matrix K and `distCoeffs` distortion coefficients.

2.2. Estimate Relative Pose

Camera pose estimation aims to determine the location and orientation of the 3D broad plane relative to the camera. Similar to camera calibration, our pipeline also utilized the implementation in AprilBoardCalibration.ipynb to extract the rotation and translation of the camera. After following the same steps as camera calibration to get `imgpoints` and `objpoints`, the pipeline utilized `cv2.findHomography` from OpenCV to find the homograph between normalized images coordinates and object coordinates. Finally, the rotation and translation were generated by decomposing the homograph.

2.3. Image Augmentation

Image augmentation was the main focus of the pipeline. Virtual objects were rendered within the image with diffuse lighting. The pipeline first rendered a transparent virtual triangular-based pyramid without lighting estimation. Then, the pipeline augmented the image with a virtual sphere with diffuse lighting. Lastly, the pipeline generated an augmented image in which multiple spheres overlap.

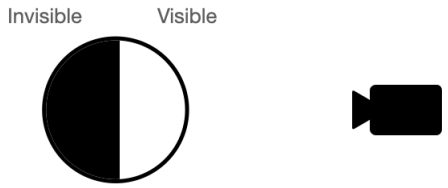

Figure 1. Our pipeline only rendered the parts of the sphere that were visible to the camera.

2.3.1 Triangular-based Pyramid

To render a triangular-based pyramid, the pipeline first defines four customized vertices (one vertex for apex, three vertices for base) in the board's coordinates. Then, utilizing `cv2.projectPoints`, the pipeline rotated and translated the 3D objects to the board's origin and projected the transformed 3D points into the camera. Lastly, the pipeline rendered the four surfaces with transparent patches.

2.3.2 Sphere

Compared to the previous task of rendering a transparent pyramid, rendering a sphere with diffuse lighting appears relatively more sophisticated. Instead of constructing a 3D sphere and directly projecting it to the camera, the pipeline only rendered the portions that should be visible to the camera (figure 1).

The following subsections would explain in detail how the pipeline localized the visible portion through back projection and minimal ray tracing. Furthermore, how the pipeline executed the lighting estimation based on the ray-tracing results.

Back Projection The pipeline localized the visible portion by back projecting the 2D image locations to the 3D locations in the camera's coordinates. See figure 2. For each pixel of the image, the pipeline utilized `cv2.undistortPoints` to remove the impact of distortion and apply the inverse of the calibration matrix K^{-1}. The pipeline then converted the results to homogeneous coordinates for the sake of the following calculation.

Minimal Ray Tracing Minimal ray tracing allowed the pipeline to calculate the ray-sphere intersection and extract the visible portion of the sphere. See figure 3.

Since the rays we generated from the back projection were in the camera's coordinates, the first step was to convert camera center O and sphere center C from the board's

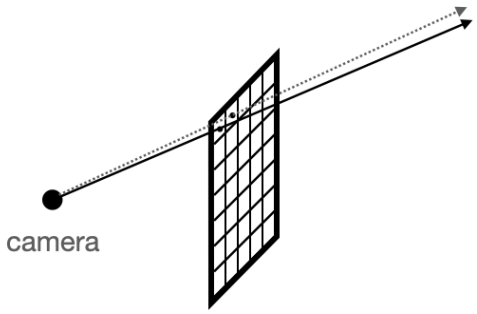

Figure 2. Back project 2D image locations to 3D locations in the camera's coordinates.

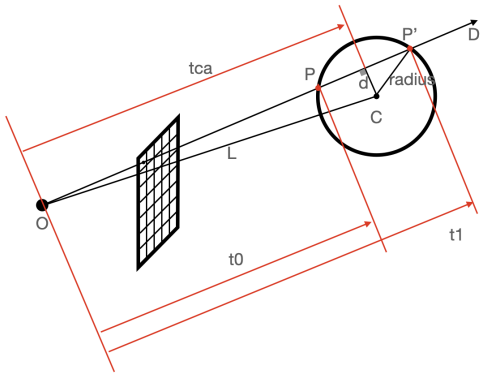

Figure 3. Ray-sphere intersection.

coordinate to the camera's coordinates by applying the extrinsic matrix.

Then, the pipeline iterated each pixel and calculated the intersection of the ray and sphere. The points on the ray could be represented as $O + t * d$. The points on the sphere could be represented as $x^2 + y^2 + z^2 = R^2$. Thus, points both on the ray and the sphere, which were the intersections, satisfied the following equation:

$$O^2 + D^2t^2 + 2O * D * tR^2 = 0$$

Note that the above equation could not be solved by computer perfectly due to loss of significance. Thus, we converted the equation to a more stable version by solving the quadratic as follows [3]:

$$q = -1/2 * (b + sign(b)\sqrt{b^2 - 4ac})$$

$$x1 = q/a$$

$$x2 = c/q$$

Lastly, for each pixel, if the ray intersected with the sphere, the pipeline selected the closest intersection and calculated its 3D location in the board's coordinates.

Lighting Estimation Given the pre-calculated intersections of the sphere and back projection vectors, we can quickly get the norm of those intersects by subtracting the sphere origin. Then, according to the simplified BDRF model, one can compute the lighting information using $a\hat{n}^T\hat{l}$, where a is assigned as the max value of grayscale and $\hat{n}^T\hat{l}$ will be normalized to the range$[0, 1]$. To simulate the real light environment as much as possible, an additional constant for ambient light is added to the $\hat{n}^T\hat{l}$ values and then normalized accordingly. Finally, assign the light information based on the pre-calculated position of the projected sphere to render a Lambertian sphere in the image.

2.3.3 Multiple Overlapping Spheres

Similar to single sphere rendering, the pipeline conducted back projection, minimal ray tracing, and lighting for each of the spheres for multiple spheres rendering. The main difference was the selection of the closest intersection. For each pixel, the pipeline not only needs to compare t among two intersections within the sphere but also needs to make a comparison with intersections generated by other spheres.

2.4. Performance Improvement

The original images captured by phone camera have a relatively high resolution, 4032 * 3024, which puts a heavy burden on calibration and geometry intersection check. In fact, the image resolution does not affect the precision of calibration much. Also, for a smooth geometry, especially a Lambertian sphere, its computed lighting values change gradually along pixels matrix, which means there's no need to calculate all the lighting information for rendering geometry. Thus, we approach performance improvement by, first, conducting all the calibration and projections based on the images with reduced size; and second, calculating the bounding box of the projected sphere in advance and slice the back projection vectors matrix before passing it to calculation.

2.4.1 High Resolution Speculation

The images are scaled to $1/4$ to the original size, which reduces the number of pixels to $1/16$. Then we implement the calibration of the camera, calculate the intersections of the sphere and back projection vectors as described before, and estimate the light based on the "small sphere." Finally, we scale the "small sphere" back to its original size, and set its lighting information as reference points, to speculate all the other pixels according to their relative distance to the reference points.

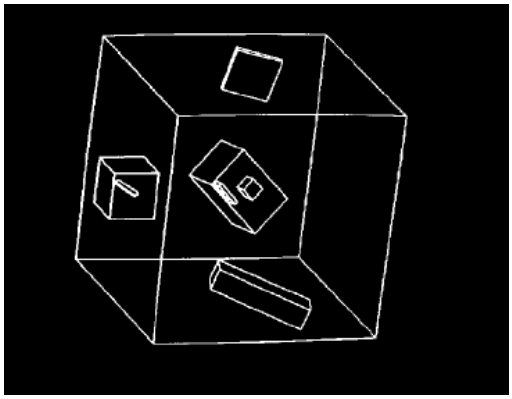

Figure 4. Arbitrarily oriented rectangular parallelepipeds modeling a hierarchically described object space.

$$\begin{bmatrix} R1 & 1/n(R2-R1)+R1 & .... & R2 \\ 1/k(R3-R1)+R1 & 1/n...+1/k... & ... & \\ ... & ... & ... & \\ R3 & & & \end{bmatrix}$$

2.4.2 Bounding Box Trim

To narrow down the calculation of intersection, we define a bounding box of the sphere to roughly estimate the possible scope of pixels where the sphere would be projected onto(figure 4). We first project the bounding box vertexes to the image and take the horizontal and vertical maximum and minimum values to slice the back projection vectors matrix. According to the size of the sphere relative to the image, the bounding box may significantly reduce the manipulation time.

3. Results

3.1. Image Augmentation

3.1.1 Triangular-based Pyramid

The pipeline rendered a transparent triangular-based pyramid with four customized data on a test image provided by the cs283 instructor team 5.

3.1.2 Sphere

Single Sphere A single sphere was first rendered on a customized image and the bounding box showed the area where the pipeline applied bounding box trim to improve efficiency 6.

Then, the pipeline estimated the lighting and rendered the sphere with diffuse lighting 7.

Multiple Sphere Two spheres with overlap with rendered on the same customized image 8.

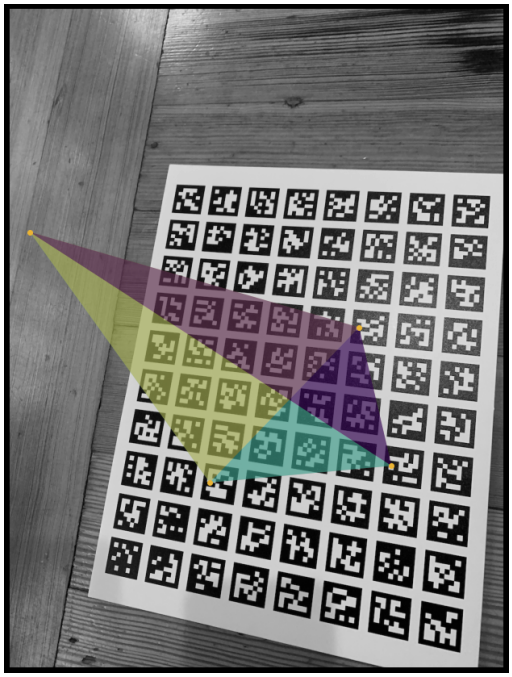

Figure 5. Render a simple AR object: Triangular-based Pyramid.

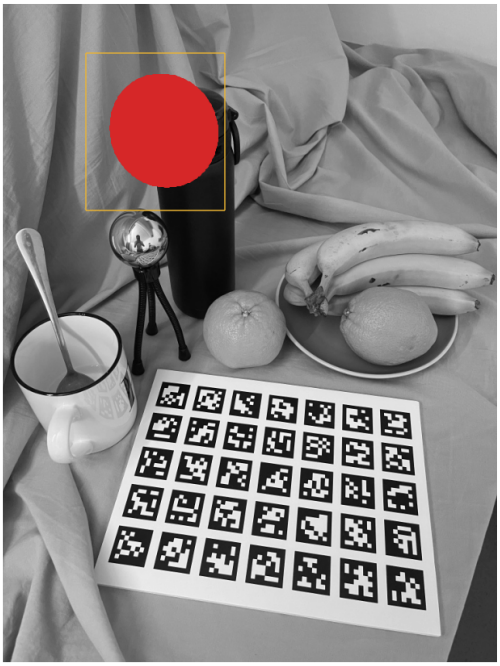

Figure 6. Render a relatively more sophisticated AR object: Sphere. The bounding box refers to the area where bounding box trim was applied.

Then, the pipeline rendered the two spheres with diffuse lighting. 9.

Figure 7. Render the sphere with diffuse lighting.

Figure 8. Render two spheres with overlap.

Figure 9. Render two spheres with overlap.

Figure 10. Control the ratio of High Resolution Speculation when scale back from the "small sphere" to original size. The left has a small ratio, can see a pretty transparent sphere. The right has a big ratio.

3.2. Performance Improvement

3.2.1 High Resolution Speculation

The pipeline scaled the "small sphere" back to the original size (figure 10).

We compared the effectiveness of high resolution speculation and found the time consumed dropped considerably after the images were compressed 11.

3.2.2 Bounding Box Trim

We evaluated how much Bounding Box Trim could improve the pipeline efficiency. Experiments were conducted

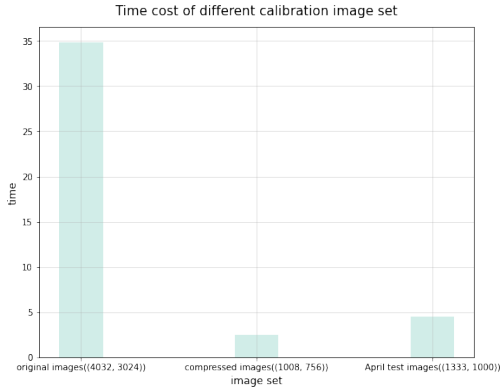

Figure 11. Time difference between different calibration image sets

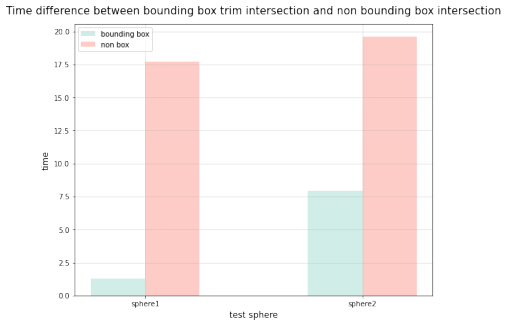

Figure 12. Time difference between bounding box trim intersection and non bounding box intersection

to measure the time difference between bounding box trim intersection and non bounding box trim intersection with different sphere locations.

Figure 12 shows that Bounding Box Trim could reduce the time consumed effectively and considerably.

4. Conclusion

The pipeline provided a feasible approach to build a simple image augmentation project from scratch without utilizing any AR tool kits. For learners with limited experience and understanding of AR development, the pipeline could provide relatively affluent support and guidance from camera calibration, pose estimation, and virtual object rendering.

However, there are still limitations on the current implementation of the pipeline that are worth further development. The lighting estimation only utilized diffuse lighting. More sophisticated lighting estimation techniques such as rendering environment lighting with the Phong reflection model would become a good extension for the project. Also, when rendering multiple objects, the pipeline excluded the impact from other virtual objects. Future development may include shadow rendering and ray tracing caused by other virtual objects.

References

[1] April robotics group. `https://april.eecs.umich.edu/wiki/Camera_suite#Calibration_targets`. (Accessed on 12/13/2021).

[2] aprilboardcalibration.ipynb. `https://canvas.harvard.edu/courses/92548/files/13552794/download?download_frd=1`. (Accessed on 12/13/2021).

[3] Minimal ray-tracer. `https://www.scratchapixel.com/lessons/3d-basic-rendering/minimal-ray-tracer-rendering-simple-shapes/ray-sphere-intersection`. (Accessed on 12/13/2021).

[4] M. Eckert, J. S. Volmerg, and C. M. Friedrich. Augmented reality in medicine: systematic and bibliographic review. *JMIR mHealth and uHealth*, 7(4):e10967, 2019.

[5] A. Fujimoto, T. Tanaka, and K. Iwata. Arts: Accelerated ray-tracing system. *IEEE Computer Graphics and Applications*, 6(4):16–26, 1986.

[6] E. Klopfer and J. Sheldon. Augmenting your own reality: Student authoring of science-based augmented reality games. *New directions for youth development*, 2010(128):85–94, 2010.

[7] A. Schollmeyer and B. Froehlich. Efficient and anti-aliased trimming for rendering large nurbs models. *IEEE transactions on visualization and computer graphics*, 25(3):1489–1498, 2018.

[8] R. Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[9] C. T. Tan and D. Soh. Augmented reality games: A review. *Proceedings of Gameon-Arabia, Eurosis*, 2010.

[10] H. Trapero, R. Ebardo, J. Catedrilla, L. Limpin, J. D. L. CUESTA, C. LEAÑO, and M. R. CHING. Using augmented reality (ar) in innovating pedagogy: Students and psychologists' perspectives. In *ICCE 2020-28th International Conference on Computers in Education, Proceedings*, volume 1, pages 87–89, 2020.

[11] A. Watson, B. Alexander, and L. Salavati. The impact of experiential augmented reality applications on fashion purchase intention. *International Journal of Retail & Distribution Management*, 2018.