Carleton Zhao
Project in Computer Vision - CS 117

Project Overview

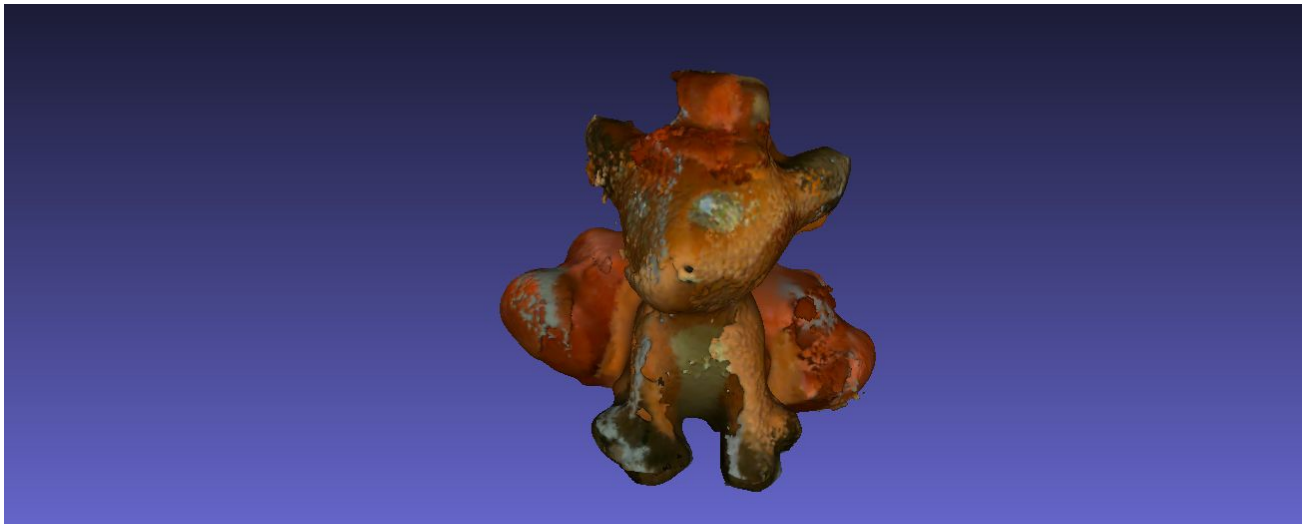
Data

Algorithms

Results

Assessment and Evaluation

3rd Party Software Used



## Project Overview

For this project I will use Structured Light Scanning to reconstruct a 3D model of a Vulpix stuffed animal. I will use multiple scans in order to accurately reconstruct a mesh of the different sides of the object. Finally I will combine all the meshes to create a solid 3D object.

My goal is to remove as many false positive points that may mess up the final 3D model. I will do this by using several different approaches before and after triangulation. I also want to align multiple meshes up and finally fill in any holes. The final model will also be colored approximately the same color as the original object.

## Data

In order to reconstruct a 3D model of the Vulpix I will be gathering 6 different scans of the object. For each scan there will be 4 calibration images and 80 structured light images. In total I will use 504 images of the Vulpix. Additionally I will be using 42 images of a chess board positioned in 21 different places to estimate the intrinsic and extrinsic properties of both cameras. These properties include focal length, principal point offset, rotation, and translation.

The setup for structure light scanning includes 2 cameras and a projector. The object that will be reconstructed is placed in front of the projector, and the 2 cameras are placed to the sides of the projector. An example of the set up is shown in *Fig. 1.*

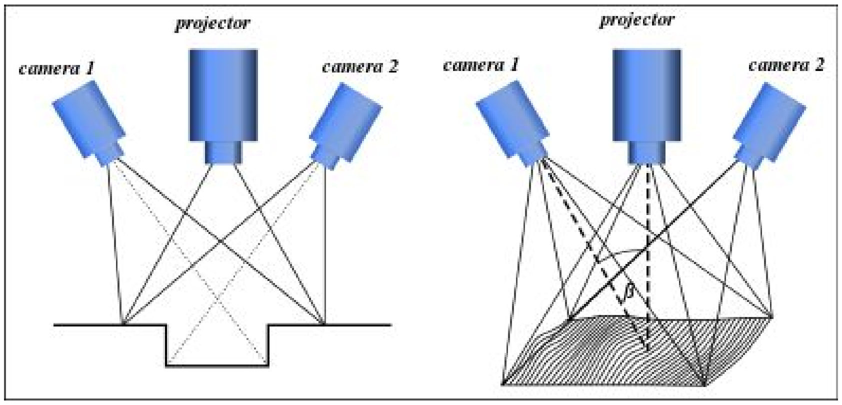


*Fig. 1*
*An example of a Structured Light Scanning setup*

https://www.wikiwand.com/en/Structured-light_3D_scanner

In each scan anywhere between 2 and 512 black and white lines are projected from the projector onto the object. The camera then takes a photo of the object and outputs a 1920px by 1200px image.

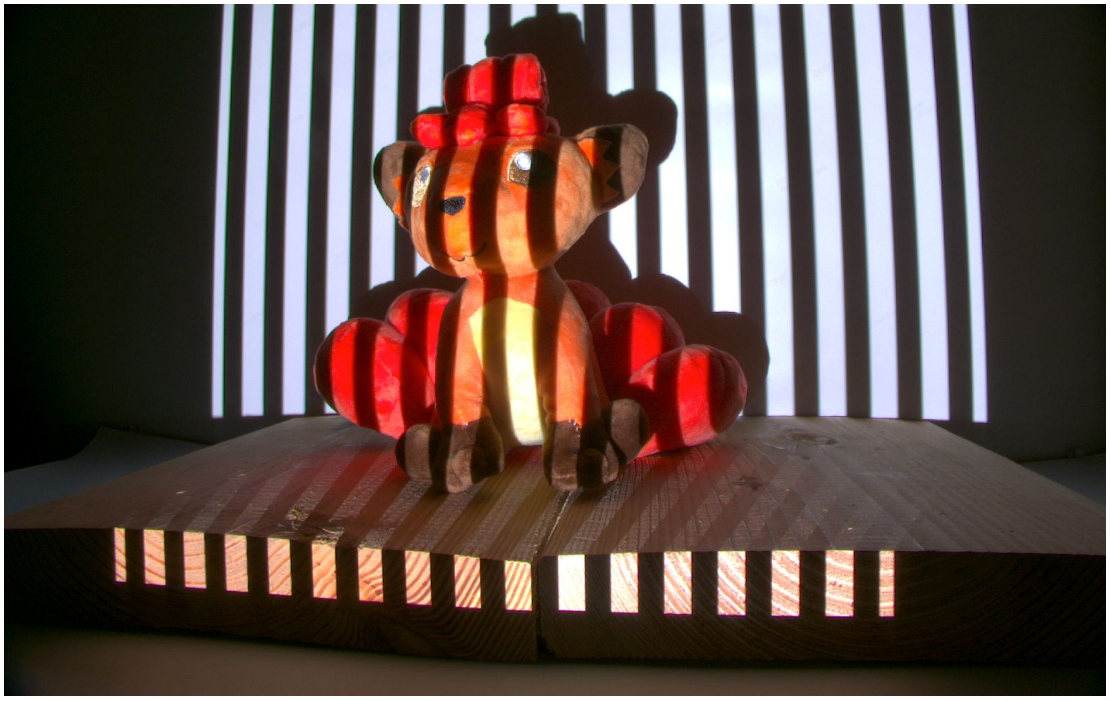


*Fig. 2*
*One of the images used to reconstruct the 3D object*

## Algorithms

### Camera Calibration: Intrinsic Properties

1. For each chess board image use OpenCV’s funciton findChessBoardCorners to find all the corners on the chess board.
2. Use OpenCV’s function calibrateCamera as well as the points found from the images of the chess board to calculate the intrinsic properties of both cameras

### Camera Calibration: Extrinsic Properties

1. Initialize a left and a right camera where focal length is the average of the x and y focal lengths. The principal point offsets are gotten from the intrinsic properties. The translation and rotation are both 0,0,0 for x, y, and z.
2. Get the pixel coordinates of the chessboard corners form two different images when the chessboard is in the same 3D location. One image should come from the right camera and the other image should come from the left camera.
3. Generate points which represent the actual 3D points of the chessboard corners.
4. Use a least squares method to minimize the difference between the actual pixel coordinates and the projected 3D points into each camera. Do this by adjusting the extrinsic parameters: x translation, y translation, z translation, x rotation, y rotation, and z rotation.

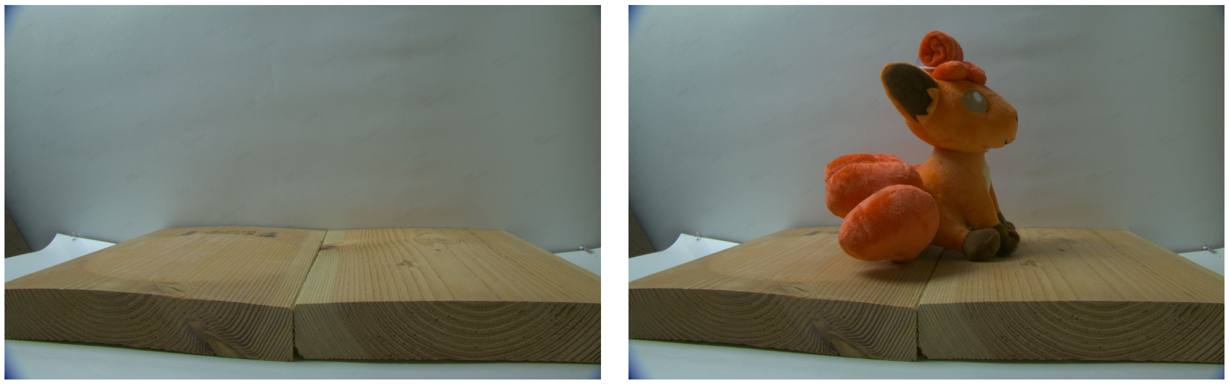


*Fig. 3*
*Example of 2 images that can be used for background subtraction*

### Background Subtraction

1. For the color value in each pixel in the first image, subtract it from the color value of the corresponding pixel in the second image. See *Fig 3* for and example of two images that can be used.

2. If the absolute value of the difference between color values is less than a threshold then replace the color value with 0. Else replace the color value with 1

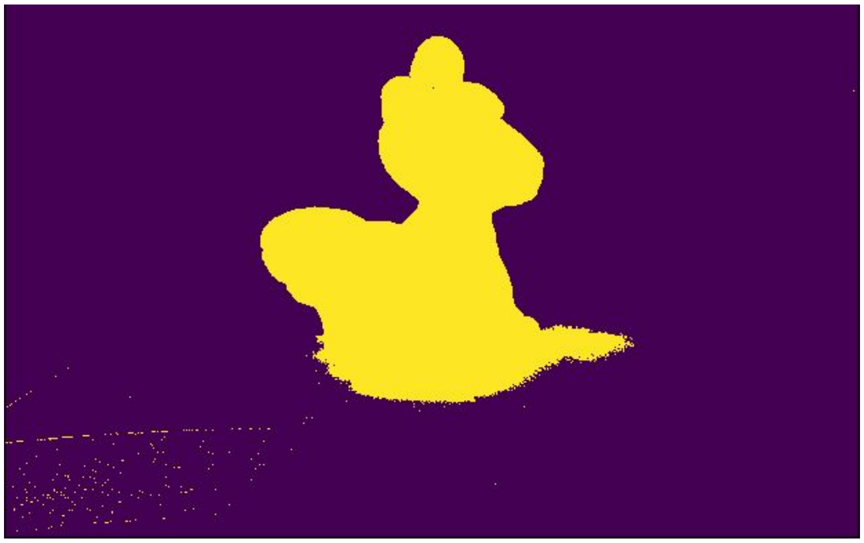


*Fig. 4*
*Example of an output for the background subtraction algorithm*

Decode Gray Code

1. For each frame and its corresponding inverse frame filter out the background using the mask gotten from the background subtraction algorithm.
2. Find the difference between the frame and the inverse frame. If the absolute value of the difference in color value of each pixel is less than some threshold set the code and mask bit for that pixel to 0. Else set the code bit to 0 for black or 1 for white
3. Use bit shifting to convert the gray code to binary and insert it into the over all code for this scan.

Reconstruct

1. Decode the specified scan for both the left and right camera using the decodeGrayCode algorithm
2. Combine the horizontal code and the vertical code for each camera
3. Combine the horizontal mask and the vertical mask for each camera
4. Find the intersection of codes from the left camera and right camera to find corresponding points
5. Using the pixel coordinates of each set of corresponding points, triangulate their positions in 3D space by using a least squares function.

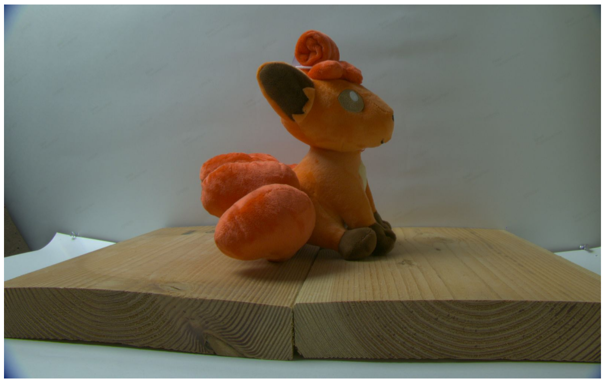
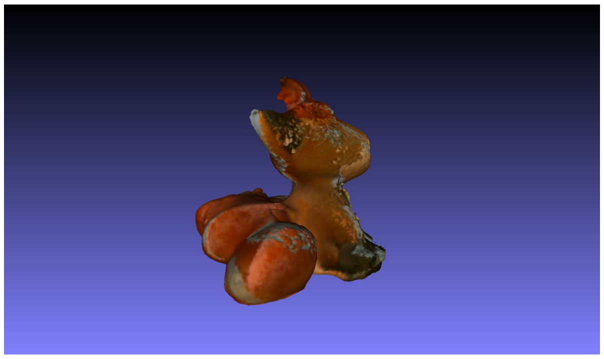
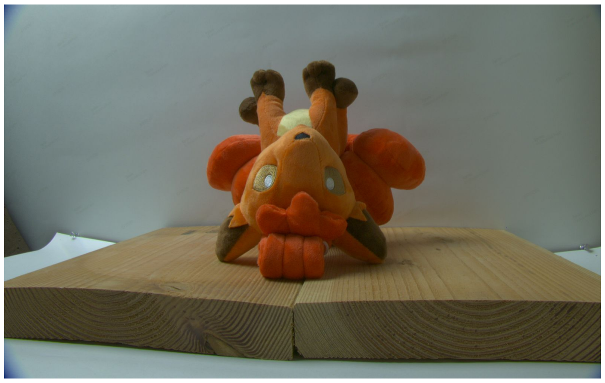
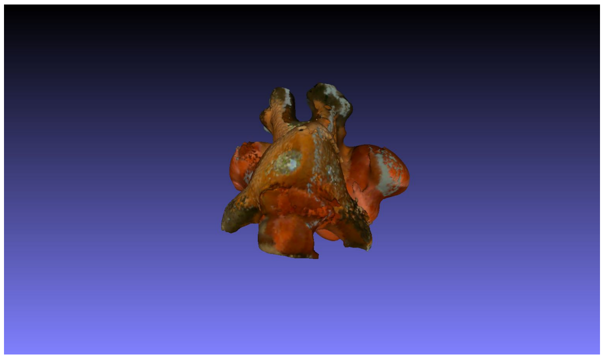
Box Pruning

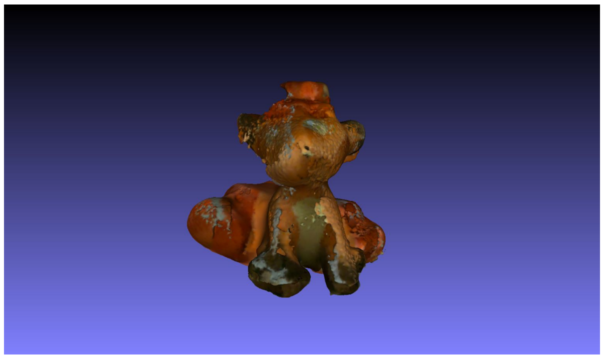
1. Remove all points that are outside a specified box in 3D space

Mesh construction

1. Triangulate all the recovered 3D points using a Delauny triangulation
2. Search through all triangles, if a triangle's edge is longer than a specified threshold delete that triangle
3. Smooth out the mesh by going through all triangles and averaging the 3D points with the other points in their corresponding triangle
4. Output all remaining triangles, 3D points, and point colors to a .ply file

## Results

| Original | Result |
|---|---|
|  |  |
|  |  |



I used meshlab to combine all the meshes that were created. After aligning all the meshes, I used a poisson reconstruction to fill in any holes that were in the mesh

## Assessment and Evaluation

Overall I am really impressed on how the final 3D model came out. Although the pipeline for this project did take a long time and definitely would not be suitable for certain applications like pathfinding in robotics, this pipeline would definitely be great for reconstruing small parts of a crime scene.

One thing I had trouble with was getting the bottom mesh to align properly. During my first few attempts at aligning the bottom mesh, the mesh would clearly not be aligned. I think I ran into this problem because the bottom mesh had very few places that aligned with the other meshes. This made it difficult to find points that should line up in meshlab. To solve this problem I used the poisson reconstruction on the rest of the aligned meshes. The poisson reconstruction was able to fill in all the holes including the one from the missing mesh at the bottom of the Vulpix. After that I realigned all the meshes, including the bottom mesh, to the output of the poisson reconstruction and did another poisson reconstruction.

## 3rd Party Software Used

Mesh Lab - http://www.meshlab.net/
Poisson Reconstruction - http://www.cs.jhu.edu/~misha/Code/PoissonRecon/Version9.01/