6.111 Final Project: 3D Scanner

Jessy Lin
Evan Tey

November 1 2016

1 Introduction

The demand for fast and cheap 3D scanning has skyrocketed since 3D printing was popularized recently on the mass market. Previously an expensive process reserved for specialized applications such as animation or industrial design, 3D scanning has expanded to day-to-day purposes such as scanning objects, people, and scenes so they can be manipulated virtually before re-printing into new models.

Potential implementations include stereoscopic imaging, photogrammetry, and triangulation from laser profiles, to name a few. The core technical challenge for most of these 3D scanners lies in reconstructing 3D data from typical 2D data, such as images from conventional cameras. This task is both complex and computationally intensive.

In this project, we will implement a fast, hardware-based version of a 3D laser scanner, using triangulation to reconstruct a point cloud for the object given the parameters for the physical setup and images from a NTSC camera. The 3D point cloud will be rendered on the display in real-time.

2 Design

2.1 Hardware setup

2.1.1 Parts Needed

- (1x + 1 extra, $4.99 ea) laser line module)
- (1x + 1 extra, $8.49/two) 28BYJ-48 stepper motor + ULN2003 driver boards
- (1x) Arduino
- Materials to build fixed laser + camera mount

2.1.2 Scanning setup

The object to be scanned will be centered on a rotating platform mounted on top of a stepper motor. It would be possible to control the motor with the FPGA to synchronize the rotation and camera capture more precisely, but as we don't foresee synchronization issues, for simplicity we will use an Arduino to control the motor rotation. The setup can thus be thought of as a rotating object (self-contained, black box to the FPGA) and the camera setup, image processor, and model renderer on the FPGA.

At regular intervals, the platform will rotate a fixed number of degrees (to be determined empirically), allowing the camera to capture uniformly spaced profiles of the object illuminated by the laser line. We will construct mount for the laser and camera that will fix their relative angles and positions so the setup does not need to be re-calibrated.

Additionally, the object will be scanned in the dark to minimize noise and increase the contrast between the laser line and the rest of the environment. This will greatly simplify later image processing.

2.2 Camera Capture

The main function of the camera setup will be to record luminance values into memory for further processing. With this hardware setup, a camera will be mounted in a fixed known location. The camera will go through an offline calibration procedure so the correct intrinsic parameters can be passed to the processing phase.

2.3 Image processing

2.3.1 Preprocessing

In order to prepare the raw camera input for the more complex algorithms further down the pipeline, we perform a series of preprocessing operations to the black-and-white camera input:

1. Noise reduction. In order to make the system more robust to camera noise, we will apply a Gaussian blur to the image. Each pixel will be recalculated as a weighted average of its neighbors.

2. Thresholding. We will use an iterative thresholding algorithm to extract the laser outline from the rest of the background based on pixel brightness. The process involves choosing an initial threshold, partitioning the image based on the threshold, calculating the mean value for each part of the partition, and recalculating the threshold accordingly.

3. Skeletonization. We will thin the outline of the laser line by applying the skeletonization structuring elements across the image repeatedly.

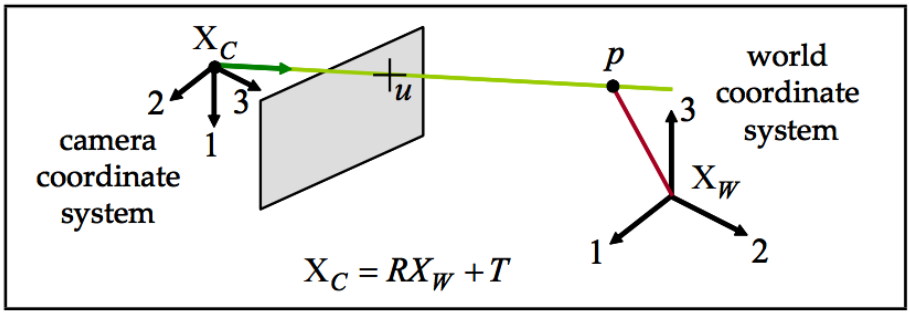

Figure 1: Figure taken from [1].

2.3.2 Depth reconstruction

Given a pixel in the processed image, we translate its image coordinates to camera coordinates by applying the transformation between the camera and world coordinates (Figure 1). We can express this in terms of the extrinsic parameters of the camera, location and orientation, as represented by a translation vector T and a rotation matrix R wrt. world coordinates. If p_C, p_W are the 3D vectors representing the point in camera and world coordinates, respectively, then

$$p_C = Rp_W + T$$

.

We model the setup as an intersection between the plane of laser light and the camera rays to the projected laser line (the object profile) to calculate p_C (Figure 2). The pixel in 2D coordinates is simply the projection of the point in the image plane.

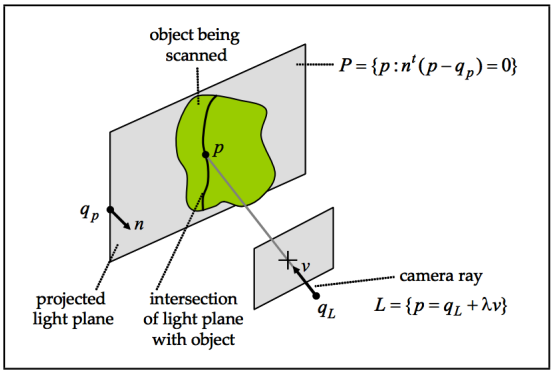

Figure 2: Figure taken from [1].

We also note that we will need to correct for the intrinsic parameters (e.g. focal length) of the camera during this stage of the process.Both the extrinsic

parameters of the camera (location and orientation with respect to the world coordinate system) and the intrinsic parameters can be determined with a one-time calibration procedure.

2.4 Rendering

Once we have a representative point cloud, we will display the the cloud on a 640x480 monitor. Modeling the screen as a virtual camera viewing the point cloud, users will be able to rotate around the cloud, viewing the object from different angles. Fixing the the user's distance, we can determine where a 3D point (x, y, z) should end up on the monitor (x', y') by a simple rotational change of bases.

3 Block Diagram

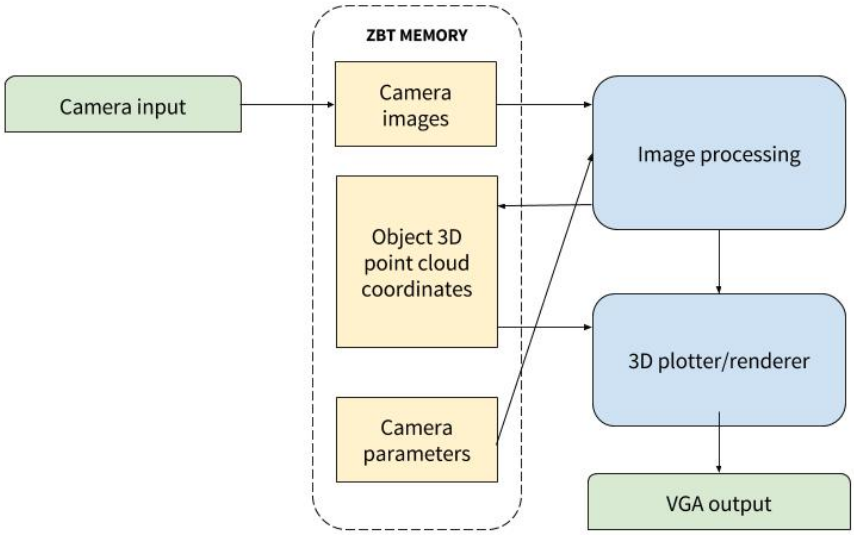

Figure 3: High-level block diagram.

4 Module Implementation

4.1 Camera capture module (Evan)

Since the main functionality of the camera module is to simply turn a video stream into luminance values, the modules will be roughly the same as the sample NTSC to ZBT files provided. These files will have to be adjusted to be

compatible with VGA transmission. This module can be tested by displaying the luminance data in the ZBT directly to the screen.

4.2 Preprocessing module (Jessy)

The filtering, thresholding, and skeletonization modules take image data from the first bank ZBT memory as input and output the processed image. They can be tested by inputting artificially generated images, displaying the output on the VGA display, and checking that they are processed as expected.

4.3 Camera calibration module (Jessy)

This module will be used for the one-time checkerboard calibration procedure. It will calculate the camera intrinsic and extrinsic parameters and output the rotation and translation matrix for the camera coordinate system, relative to world coordinates.

4.4 Depth reconstruction module (Jessy)

The depth reconstruction module will take camera transformation and rotation matrices (from the previous module) and the processed image (from ZBT memory) as input. It will output a set of 3D coordinates to the second bank of ZBT memory, to be processed by the renderer. Testing this module will require checking that, given 2D pixel locations for which we know the 3D coordinates (e.g. the center of the world coordinate system at the center of the scanner, the camera itself), the coordinates are indeed calculated correctly.

4.5 3D rendering module (Evan)

The main module in the 3D renderer will take in the camera position as well as data from the point cloud ZBT and transform the data into 2D points to be displayed in the monitor. This means first, a virtual camera module is needed to specify the (θ, ϕ) position of the virtual camera in response to user input (up and down buttons), and a VGA display module is needed much like in lab 3 to display the points generated from the transformation in the main module. Since this won't be fast enough to be directly displayed, these transformed points will be passed into a buffer to be displayed during the next cycle.

This can be tested iteratively, starting with simply rendering a self generated set of points (like a cube) from a fixed position (maybe even before this, a serial connection can be used to test the points generated by the main 3D rendering module in a Matlab plot). Next, the virtual camera can be tested on the point cube, and last, the ZBT timing integration can be tested via testbench.

5 Stretch Goals

The main stretch goal we have is creating a mesh from the point cloud and rendering that mesh. A variety of surface generation (for example, Delaunay triangulation) exists to generate surfaces from point clouds. These polygons would then need to be stored (likely via their vertices), then rotationally transformed according to the virtual camera, and finally shaded with some basic shading.

Separate stretch goals include allowing non-fixed virtual camera distance from the object (so users could zoom in and out) or gesture-controlled virtual camera movements.

For the camera capture / image processing sections, a stretch goal is line extraction without darkness (so, recording color data for the camera and performing hue filtering to select the laser).

6 References

[1] http://mesh.brown.edu/byo3d/notes/byo3D.pdf