

CS635: Assignment #1 – Real-World 3D Reconstruction

Out: January 26, 2007, 10:21am

Due: February 9, 2007, 9:29am

Objective

Extending the previous assignment, the objective of this assignment is to replace the synthetic data with real-world data and to reconstruct a simple 3D object. First calibrate either a course-provided digital camera or one of your own. Then, “click” on corresponded scene points and on known scene-point fiducials (to estimate pose). Finally, reconstruct the observed scene points and project and blend the captured images onto the recovered model.

Camera calibration was explained in class. In this assignment, you do not need to implement the calibration software from scratch but rather will use an existing implementation of the Tsai camera model. You will need to gather the calibration data, perform the calibration, and verify the results. Then, you will have to extend your software system to render real reconstructed-data and not synthetic reconstructed data.

This assignment requires some mental planning work, some lab work, and some programming. The whole assignment is not that much work and mostly requires you to understand the process. I do recommend STARTING EARLY! (i.e. -- you will not be able to complete the assignment if you start the day before its due --).

Detailed Description

Step 0 – Internal Parameters Calibration

In this assignment, you will calibrate a digital CCD camera (all 5 internal and 6 external parameters). You may use a camera I provided you with or you may use your own digital camera. If you use your camera, you’ll need to know some manufacturer specifications such as the actual resolution of the camera and the actual size of each pixel. These are typically found in the camera’s manual.

To perform the calibration, you will need a calibration pad. There are two parts to the calibration phase. First you calibrate the internal parameters (once). Then you must discover the external parameters each time you move the camera and take a picture. You may fabricate your own calibration pad or use one I’ll provide in the lab. Precision here is of the utmost importance. The calibration pad in the lab is fairly large (which is good), quite flat, and quite precisely printed (it costs about \$500 to make so please treat it carefully) – this pad STAYS in the lab.

To calibrate the internal parameters, you will probably only need to take one picture and observe at least 11 points for the software package I provide (but the more points, the better). Details on the software calibration package are at the end of the assignment.

Step 1 – External Parameters Calibration

To determine the external parameters (pose) you will fix the internal parameters and re-estimate the pose each time you move the camera and take a picture. You can use the calibration pad to do this work or a set of fiducials/markers you place in the environment and measure their relative 3D

location. I have placed a set of light-boxes in the lab -- you can use these (take a few, bring them back later), make up your own markers (anything which you can clearly see and precisely click on is good). If you use markers, you will have to measure their relative locations.

From these observations you can estimate external parameters. The calibration software has functions to do this and/or you can do it yourself via a simple function minimization (e.g., nonlinear-least-squares, e.g., Levenberg-Marquardt).

Now, you have calibrated internal parameters (fixed for the camera) and calibrated external parameters (a full set needed for each picture you take).

Step 2 – Point Reconstruction

The next step is you place your calibrated camera and your images into the previous assignment and reconstruct some scene points. Using at least two calibrated views, click on corresponding points in the two images and reconstruct their 3D location. The more points you correspond, the more 3D data you have. Draw the reconstructed point as a small dot on top of the image. You can use basicmodeler/OpenGL do all of this.

Step 3 – Triangulation and Texture-Mapping

To demonstrate the system, please choose an object of your liking, take at least two pictures (e.g., put the object on a table, put the markers/pad around it, take two pictures from about 30 to 60 degrees apart), and then show a texture-mapped reconstruction. You will have to click (manually) on sufficient points so as to capture the general structure of the object. Then, you will have to triangulate the points (in 2D – a routine to do this is provided, it does a ‘Delaunay Triangulation’). Finally, map each of the two textures onto the triangulation, render both textures simultaneously, and blend them (see `glBlend()`).

Demonstration

Your demonstration will consist of showing me a simple 3D model of your reconstructed object. Create the points/geometry using the “BasicModel” object class and then you’ll be able to save/load the object just like the *.sim test objects you had for the previous assignment. (Note: if you **really really** want to create your own object format, please ask the instructor). Thus, you can simply provide me with the model and the images and I can view it on my version of basicmodeler.

Using only two images and a few corresponded points, the blended images will not necessarily produce a very nice looking object. We will work on improving the quality later. You can, however, improve the results by choosing you object wisely, changing the angular difference between images, and CALIBRATING YOUR CAMERA WELL.

Grading and Deliverable

Your grade will be influenced by how well your particular object is reconstructed, by the presentation and usability of your program, and by how well you complete the assignment requirements.

I will schedule a brief meeting with each of you and you can demo your assignment to me. A zipped file containing source, compiled executable, and auxiliary files must be emailed to me before the due date.

Tsai Software

The Tsai software is located in

<http://www.cs.purdue.edu/~aliaga/cs635-07/tsai.zip>

It is currently compiled under Win32. To compile, use Cygwin and make a library makefile containing all *.ch files (except for tsai_demo.c) or make a MSVC library project and add all *.Ch files to the project (except for tsai_demo.c).

The file tsai_demo.c is an example top-level calibration program using non-coplanar points (as might be the case with this assignment; if not use coplanar_XXXX). Take a look at it. You should also look at tsai.h – the main interface to the Tsai calibration software. The Tsai library does rather complicated stuff but you really only need to look at tsai_demo.c and tsai.h. That program reads tuples from stdin (e.g., test.cd) and outputs the camera parameters to stdout (note: the Tsai class does support reading/writing to a file). It also projects 3D points to 2D pixels. You could modify tsai_demo and use the program as a system call or you could incorporate the functionality into your program and link with the Tsai code (preferred method, because of flexibility).

Briefly the most relevant Tsai object class methods are:

```
[non]coplanar_calibration_with_full_optimization()
    Call this function to calibrate all parameters

[non]coplanar_extrinsic_parameter_estimation()
    Call this function to calibrate external parameters

world_coord_to_image_coord()
    Call this function to project from 3D to 2D
```

You will also have to setup the Tsai.cp public variables according to your camera and image resolution – it's pretty straightforward. If you have questions about this, just see me.

The Tsai calibration library works best if the world origin is not near the middle of the field of view and is not near the y-axis of the camera coordinate frame. One way to ensure this is to temporarily offset the calibration data points by 1000 cms (e.g., very far away), calibrate, and then undo the offset.

Triangulation

I have placed in <http://www.cs.purdue.edu/~aliaga/cs635-07/triangulation.zip> a copy of a Delaunay triangulation package. This software constructs the Voronoi region of a set of 2D points, computes the dual of that (called a "Delaunay triangulation") and returns a triangulation where all triangles are roughly the same size. The original makefile compiles the code using GNUmake (I have not included it). You'll have to write a new makefile or just incorporate the code into your project – it's pretty straightforward. Just compile everything with the environment variable LIBRARY defined. If you want to use another triangulation package, go ahead. To use

this code, just include “fortunedelaunay.h” and call the function
delaunay_triangulation(float *points, int n, int **tris, int *ntris).
The points array contains a list of ‘n’ 2D vertices and the tris array contains ‘ntris’ triples of
vertex indices that form the computed triangulation. This code is an adaptation of freeware
written by a well known researcher, Steve Fortune.

Collaboration

In this assignment, you may collaborate only to help with the mechanics of the assignment (e.g.,
using Tsai, triangulation, etc). *Everybody must take their own pictures!* Practically speaking, this
means that nobody should have the same pictures or calibration results.

If you have questions, please come see me ASAP – do not wait until the last moment.

Have fun and good luck – you’re on your way to capturing a 3D model!