Visualization of Cultural Heritage

Velu Ganapathy
NiCT, Tokyo, Japan
Summer 2010
Who am I?

- 2nd year BS Computer Science student at University of California, San Diego
- UCSD Mentor: Jurgen Schulze
- Host Mentor: Shinji Shimojo
Proposed Research

- The goal of our collaborative project is to visualize cultural artifacts on a tiled display wall (TDW)
- We hope to incorporate camera-based user tracking so that the perspective of the 3D object will change in accordance to where the audience is located
My Component

- Responsible for generating the content that Lex would interact with
- Base off the existing Bundler algorithm and improve to accommodate stereo image pairs
What is Bundler?

- “Bundler is a structure-from-motion system for...image collections...written in C and C++”
- “Bundler takes a set of images, image features, and image matches as input, and produces a 3D reconstruction of camera and (sparse) scene geometry as output”
- Similar to Microsoft Photosynth
Motivation for this Project

- **Stereo Image Pairs**
  - Generate a better, more detailed point cloud

- **SURF over SIFT**
  - Speeded Up Robust Features
  - Scale Invariant Feature Transform
  - Faster and more robust

- **Openness**
  - “In-house”, “done at UCSD”
  - Greater understanding of internal workings
  - Easier to modify/change
Initial debate: CGLX vs. COVISE

- Both are options to interact with TDWs
  - CGLX – Falko Kuester
  - COVISE – Jurgen Schulze
- Decided to use COVISE
  - Sasha Koruga’s PhotosynthVR plugin
    - “in-house”, “done at UCSD”
What I Do (in a nutshell)

- Calculate certain parameters and values and print output in a certain format so that PhotosynthVR plugin can handle it
- What format? Bundler format
What I Need

- I need to generate / calculate / provide the following values and parameters

- Camera entry
  - \(<f> <k_1> <k_2>\) [the focal length, followed by two radial distortion coeffs]
  - \(<R>\) [a 3x3 matrix representing the camera rotation]
  - \(<t>\) [a 3-vector describing the camera translation]
What I Need (cont.)

- Point Entry
  - <position> [a 3-vector describing the 3D position of the point]
  - <color> [a 3-vector describing the RGB color of the point]
  - <view list> [a list of views the point is visible in]
How I Do This

• The crux of my work involves calculating:
  ◦ 4x4 matrices (the <R> and <t> values mentioned earlier)
  ◦ positions of the 3D points
OpenCV

- Utilize OpenCV, a C++ library for computer vision
- Use cvExtractSURF method to extract Speeded Up Robust Features from an image
- Find correspondences between two images of the same object
There are 2 stereo image pairs, L1 R1 L2 R2
Explanation

- I am able to say, "this feature point in image1 \((x_1, y_1)\) corresponds to that feature point in image2 \((x_2, y_2)\) and the associated 3D point (using cvTriangulatePoints) is \((x, y, z)\)"

- The 4x4 matrix is “the red arrow” that correlates the two views, from one coordinate system to another
Explanation (cont.)

- I use C++'s std::map functionality to find the 3D points associated with the points that match in 2D (R1 and R2)
- **NOTE:** R1-R2 is arbitrarily chosen, could have also done L1-L2
The 4x4 Matrix

• I isolate 4 of those 3D points to create the 4x4 matrix:
  \[
  \begin{bmatrix}
  x_1 & x_2 & x_3 & x_4; \\
  y_1 & y_2 & y_3 & y_4; \\
  z_1 & z_2 & z_3 & z_4; \\
  1 & 1 & 1 & 1
  \end{bmatrix}
  \]

  for both the views P and P’

  \(<x, y, z, 1>\) is a position vector

  And solve for the transformation matrix \(X\):

  \[
  X^*P=P’
  \]

  \[
  X = P’ * \text{inv}(P)
  \]

• I then apply the transformation matrix to all the 3D points in P to determine their location in the coordinate system of P’

• So now points should be in same coordinate system?
Reprojection Error

- **BUT**, this is not a rigid transformation!
- Objects will be skewed a little due to errors stemming from:
  - Imperfect projection model
  - Inaccurate camera calibration
  - Inaccurate 2D feature coords in image space
  - Numerical errors
  - Etc.
- Matrix will transform a point to another location correctly for only the four points used in the calculation but not for an arbitrary case...
Solution: Orthonormalized Matrix

- I take the rotation part (3x3) as \([XYZ]\) (making sure \(|X| = |Y| = |Z| = 1\)) and make a new \(Z' = X \times Y\), then \(Y' = Z' \times X\), to get \([X,Y' Z']\)

- I finally have the values I need
  - \(<R>\) [a 3x3 matrix representing the camera rotation]
  - \(<t>\) [a 3-vector describing the camera translation]
  - positions of the 3D points
Camera Calibration Matrices

- To get “<f> <k1> <k2> [the focal length, followed by two radial distortion coeffs]” I calculate the camera intrinsics and extrinsics.
- Utilize the provided OpenCV sample code, stereo_calib.cpp along with photographs taken from my Fuji W1 stereo camera.
  - Use StereoPhoto Maker to break up MPO file format into left and right side images.
Problems

- Unfortunately, due to errors in logic and/or in my code, I continued to experience issues in displaying the point cloud correctly
  - Outliers...
- I need to consider “bundle adjustment” to find a matrix that minimizes the reprojection error
Nara Festival

- **FALLBACK PLAN**
  - We resort to using Bundler to create bundle.out, which is fed to PhotosynthVR plugin to display on TDW
  - Will work with Professor Jurgen Schulze upon my return to get algorithm to work correctly
Acknowledgments

- University of California, San Diego, PRIME, Calit2
  - Professor Jurgen Schulze
  - Dr. Peter Arzberger
  - Dr. Gabriele Wienhausen
  - Teri Simas
  - Jim Galvin
  - Tricia Taylor-Oliveira
  - Haili Wang, Phi Nguyen, Sasha Koruga

- NiCT (Koganei/Otemachi/Keihanna/Osaka)
  - Professor Shinji Shimojo
  - Yoshinori Kobayashi
  - Masaki Chikama
  - Professor Kiyoshi Kiyokawa