Deriving Camera and Point Location From a Series of Photos Using Numerical Optimization

by Chris Studholme

Abstract

The goal of this project is to discover what attributes of a 3 dimensional scene can be derived from a series of ordinary photographs of that scene. In particular, given a series of photographs of a scene taken from arbitrary and unknown locations, is it possible to determine the precise location and orientation in 3D of the camera when each of the photographs was taken. The answer appears to be yes, although the technique is not trivial. Presented here is an algorithm to obtain the necessary information using a numerical optimization technique known as differential evolution. In addition to finding the location of the camera for each photo, several points known to be on the surface of the object being photographed are also located in 3D. The technique has been applied to photographs of two simple household objects and, with somewhat less success, to a series of photos of an entire room. Some ideas and future directions for recovering more of the 3D scene are also discussed.

Introduction

Mesh acquisition and surface reconstruction are two important topics in computational geometry. Generally, the goal of surface reconstruction is to find an approximation to some unknown surface given a set of points sampled from that surface, or a mesh. Mesh acquisition is the process by which these points, or mesh, are sampled. Most often, meshes are acquired using some sort of device that "touches" the surface of the object to sample points on that surface. Laser and radar range finders do this by bouncing a laser or radar beam off the object to compute the distance to the surface. An even more direct means of sampling a surface is by using some sort of mechanical probe.

The goal of my geometry project is to sample an unknown surface in a less direct way. By using only an ordinary film or digital compact camera and not even so much as a tripod to keep track of the location of the camera, my goal is to obtain an approximation of some unknown surface from a series of photos of that surface. The first step towards this goal is to compute the location and orientation of the camera when each of the photos were taken. To this end, I also compute the location of one or two dozen points on the surface of the object being photographed. After the camera has been successfully located for each photo, a variety of techniques can be applied to recover the unknown surface. Described in this paper is the technique I have used to derive the location and orientation of the camera when each of the photos was taken and to locate several points on the surface of the object. This paper does not describe a technique for recovering any more of the surface than the few points found; however, several techniques that may work are suggested.

Outline

The technique described here has many applications, not just in terms of traditional mesh acquisition and surface reconstruction, but also in areas such as vision and other related fields. In the next section I will describe some of the previous work in this area. After that, I discuss the approach I have taken towards solving the problem and the numerical techniques I have employed. Then, the various pieces of software I have implemented to mark pixels in the photos, derive the camera and point locations, and to visualize the results are presented in detail. Finally, in the conclusions section, I mention some of the difficulties associated with this approach and describe various possible directions one could go in to obtain a more complete sampling of the unknown scene.

Previous Work

Kutulakos and Seitz have developed a theory of shape by space carving [Kutulakos 1998]. Their technique aims to reconstruct an arbitrary scene from a series of photos taken from arbitrary, but known locations. Kutulakos has also done some more recent work [Kutulakos 2000] on the problem of determining structure from a series of photos taken from unknown locations. This work has been inspirational to me not necessarily for teaching me any particular techniques to use, but for showing me that my goal is feasible.

Deriving 3D structure from a series of photos is not without its problems. In general, it is not possible to find a unique structure capable of explaining the observed photos. Kutulakos and Seitz describe an "equivalence class of all 3D shapes that reproduce the input photographs" and provide a technique for finding one member of that equivalence class, the *photo hull*. For the purposes of this project, I am content with finding any one solution consistent with the input photos.

Approach

The approach I have taken for this project is as follows. First, I took several photographs of a few sample objects. One set of photos was of a small stuffed penguin, another was of a cordless trackball, and the third set of photos was of a small room. The camera I used for this project is a Nikon CoolPix 950 digital camera. The camera is capable of producing photographs with a resolution of 1600x1200 which, when JPEG compressed at the highest quality setting (least compression), produces files that are approximately 800 kilobytes in size. The camera has a 1x to 3x zoom feature, but for the purposes of this project, the zoom setting was not varied between photos of a particular subject. For the photos of the small room, a 0.45x wide−angle lens was used. The camera has a feature that allows the white balance and exposure settings to be locked for a series of photos. This feature was used to increase the consistency between photos which I assume would be required (or at least desired) if the general problem of finding 3D shape is to be solved. For each subject, I took 6 to 9 photos; each of which were taken in a very *ad−hoc* manner with no attempt made to measure the location, direction or orientation of the camera. As mentioned, the focal length (zoom setting) of the camera is the only camera parameter assumed to be constant across photos.

I have assumed that my camera can be modeled as simple pin−hole camera. The following parameters describe the camera at the moment a photo is taken: the position of the focal point (the pin−hole) of the camera in 3 dimensional space, a direction vector indicating which

direction the camera is pointed, an up vector indicating the orientation of the camera (which direction is up), and the focal length of the camera (distance between the focal point and the image plane). There are clearly 3 degrees of freedom in describing the position of the camera; however, since the direction vector can be fixed to unit length, there are only 2 degrees of freedom in its choice, and since the up vector can be fixed to both be unit length and be orthogonal to the direction vector, there is only one degree of freedom in its choice. This implies that there are at most 7 degrees of freedom in the position and orientation of the camera for each photo. For the purposes of my project, the focal length was held constant across all photos of a single subject, and therefore, I only have to consider 6 degrees of freedom for each photo.

The equation used to cast a ray from a camera position, through a pixel in the image plane, to the location where the point represented by that pixel is assumed to be located is given by:

$$\vec{proj} = fl \cdot \vec{dir} + x \cdot pw \cdot (\vec{dir} \times \vec{up}) + y \cdot pw \cdot \vec{up} \tag{1}$$

where $\vec{dir}$ is unit length, $\vec{up}$ is unit length and orthogonal to $\vec{dir}$, x and y are the location of the pixel in question, and pw is the width of the pixel in the image plane. Since the magnitude of $\vec{proj}$ is irrelevant, pw and fl are not independent. For the purposes of this projection, only the ratio of pw to fl needs to be specified. The pixel width parameter is a property of the camera that should be known; however, for my camera I don't know the value of that parameter. Furthermore, although I kept the focal length fixed throughout each set of photos, I don't know its value either. Therefore, I decided to have my numerical optimization fit for the ratio of these two parameters in addition to the locations of the camera.

The inverse of the above equation gives the location of the pixel in the image plane that corresponds to a given point in 3D. It is given by:

$$\begin{aligned} x &= \frac{fl}{pw} \cdot \frac{(\vec{point} - \vec{pos}) \cdot (\vec{dir} \times \vec{up})}{(\vec{point} - \vec{pos}) \cdot \vec{dir}} \\ y &= \frac{fl}{pw} \cdot \frac{(\vec{point} - \vec{pos}) \cdot \vec{up}}{(\vec{point} - \vec{pos}) \cdot \vec{dir}} \end{aligned} \tag{2}$$

where $\vec{point}$ is a point in 3D visible in the photo and $\vec{pos}$ is the location of the camera. Notice that pw and fl are again not independent and that only the ratio of these two parameters needs to be known.

To make use of the above equations I needed to either know the position in 3D of a variety of points or the location of the pixel within each photo that projects to the points in 3D. The latter is easily obtained. After taking a series of photos as described above, I decide on a set of easily recognizable points on the surface of the subject in the photos. These points were typically sharp edges in the structure or texture of the object. After the set of points has been decided on, it is fairly easy to go through each of the photos and indicate to the computer with the use of a standard mouse the location of each of the points within each photo. Obviously, each point does not necessarily appear in every photo as some with be on the "back side" of the object with respect to that photo. My goal was to have each point appear in at least 3 of the photos, but as few as 2 is fine and more is better. The software I used to manage this process of marking points is a software package designed for creating web−based Internet

photo albums. This software has a feature that allows the creator of the album to maintain a database of people who might appear in the photos and to indicate where in each photo each person may appear. This feature of this software is well suited to my task. The only problem with using this software is that there is no means to zoom in to a particular location in the photo to mark a pixel with the greatest accuracy. Because of this deficiency, the pixels I have identified may be off from the correct pixel by as many as 2 or 3 pixels.

To derive the location in 3D of each of these surface points and to determine the location and orientation of the camera when each photo was taken, I have cast the problem as one of numerical optimization. The error function is pretty simple. For each point that appears in a given photo, compute the pixel that the point should appear in using equation 2 above and then sum up the squares of the distance between the computed pixel and the actual pixel the point appears in (as marked by the user).

One important question that must be answered here is "how many points on the surface of the subject need to be specified?" To ensure that the optimization problem has a unique solution (or at least finitely many) I need to ensure that the (non−linear) system of equations it aims to solve is overstated. To be sure of this, I need to count the number of degrees of freedom (free variables) and the number of associated constraints.

As mentioned earlier, the number of degrees of freedom associated with an individual photo is either 6 or 7 depending on whether the focal length can vary or not. The number of degrees of freedom associated with a point on the surface of the subject is 3. Now, letting the position and orientation of the camera for all of the photos and the position of all points vary freely will not yield a unique solution as the coordinate system I am going to use to define these positions and orientations needs to be specified in some way. To define this coordinate system, I need to restrict some of the camera positions or points (or both). For the sake of this analysis, let me fix the position of the camera for the first photo to be at the origin of the coordinate system and let the direction and orientation of this camera define the orientation of the system. The focal length of the camera for this first photo will define the scale of the coordinate system. Finally, if the width parameter (pixel width divided by focal length) is allowed to vary, then the distance between the origin and at least one of the surface points also has to be fixed. For the sake of simplifying the equations a little and since I only seek a lower bound on the number of points, I'll ignore this last restriction for now. The number of free variables is thus given by:

$$variables = 7(m-1)+3n \tag{3}$$

where m is the number of photos and n is the number of surface points. Note that if focal length is fixed, then the 7 should be changed to a 6.

Each observation of a surface point in a photo (a pixel) can be seen as 2 constraints: one for the x coordinate and one for the y coordinate. To get some idea of how many surface points may be needed, I will assume that each surface point appears in at least half of the photos, or equivalently, I'll assume that each photo has at least half of the surface points visible within. If this is the case, then the number of constraints is given by:

$$constraints = 2\frac{n \cdot m}{2} = n \cdot m \tag{4}$$

By equating 3 and 4, the minimum number of surface points can be determined with:

$$n = \frac{7(m-1)}{m-3} \tag{5}$$

With 6 photos, at least 12 points are required. With 9 photos, 9 points are required. In general, this function decreases monotonically as the number of photos increases and has a limit value of 7 (6 if focal length is fixed). With all of the subjects I tested, I had between 14 and 21 surface points, and thus was above this limit; although, each surface point I defined didn't appear in half of the photos so I had to verify the feasibility of my computations by directly counting variables and constraints. Note that there are some other restrictions as well. Since each observation provides 2 constraints and each photo has 6 or 7 free variables, there must be at least 3 or 4 observations per photo. Also, each point has to appear in at least 2 photos to have any hope of being located it in 3D. Equation 5 seems to imply that there must be at least 4 photos. This is just an artifact of the assumption that each point appears in only half of the photos. Considering most animal's (including human's) ability to derive 3D shape from only two views of a subject, the minimum number of photos required is actually 2. With only two photos, all points would have to appear in both photos.

I found that if I defined my coordinate system as described above, that my optimization program would always find the solution where the camera for all of the photos and all of the points are located at the origin. Since this solution has an error of zero, it is hard to blame the optimization algorithm for finding it. To avoid this solution, I had to define my coordinate system in a different way from what was described above. I decided to choose two of the surface points and put them at the locations *(0,0,0)* and *(0,0,−1)*. Furthermore, I restricted the camera for one of the photos to the plane defined by $y=0$ and $x<0$ and insisted that it be oriented such that these two fixed points are projected perfectly within that photo. I'll describe this in a little more detail later in the implementation section of this paper.

My first attempt at solving this optimization problem was with the use of the Simplex algorithm as described in [Caceci 1984]. This failed miserably as I discovered that the surface defined by my error function is very complex and has countless local minimum that the Simplex algorithm had no hope of avoiding. I decided to instead apply a genetic algorithm called Differential Evolution as described by Price and Storn [Price 1997]. This algorithm is described as "a simple evolution strategy for fast optimization." Let me briefly describe how it works. The algorithm maintains a population of possible solutions to the problem and the value of the error function associated with each member of the population. Then, during each *generation*, each member of the population gets a chance to play the role of a parent, the target vector, and is mated with some other randomly selected and mutated member of the population for the purposes of producing offspring, a test vector. The error function is then evaluated for this test vector and the test vector is put into competition against the target vector. If the test vector has a lower error value, it replaces the target vector in the population; otherwise, it is discarded and the target vector remains as is. To obtain an appropriate mutation to apply to the randomly selected parent, two other randomly selected population members are differenced and then scaled by a user specified parameter. Another user specified parameter controls the cross−over of parameters between parents to generate the child. Although somewhat finicky and obviously nondeterministic, this algorithm has proved successful for this type of optimization problem.

Finally, to observe and analyze the output of the optimization problem, I wrote a Java3D program to plot the position of the camera for each photo, the surface points, the rays projected through each photo, and the convex hull of the points. The program allows user input via the keyboard to vary the viewing angle and visible features.

Implementation Details

As described above, the surface points are identified using software that was not written specifically for this project, and therefore, will not be discussed here further. More information about this software can be found at: http://vphotoalb.sourceforge.net/. The output of the software is a plain ASCII file that describes the surface points identified, the photos, and each observation of a point in a photo. These items appear in three sections like so:

```
1 Left Eye
2 Right Eye
  ...
16 Dot on back of Head

6 tux/dscn2627t.jpg 1600 1200
7 tux/dscn2628t.jpg 1600 1200
  ...
8 tux/dscn2634t.jpg 1600 1200

1 1 0.4331 0.2883
1 4 0.6 0.32
  ...
16 3 0.4356 0.1583
```

Both the implementation of the optimization problem and the program to view the results were implemented using the Java programming environment. The latter program also requires the Java3D programming library.

To aid in the evaluation of the geometry equations that must be solved, a class named NVector was created to represent a vector or a point in 2 or 3 dimensions. This class supports all of the basic vector computations such as addition, scalar multiplication, inner product, cross product, normalization, etc.

To solve the optimization problem using the method of differential evolution, a general purpose package consisting of a set of interfaces for parameters and observed data, along with a class named DE implementing the algorithm was written.

As described above, differential evolution works by crossing parameters from one parent with a mutated version of the parameters from another parent. One approach that I could have taken would have been to have each individual free variable represented as a parameter. If this were done then it would be possible for a particular photo, say, to have the x and z coordinates of the camera's position come from one parent, while the y coordinate comes from the other parent. This seemed to me to be a rather unnatural crossing of the parameters. A more natural crossing would be to have the camera position and orientation for one photo come from one parent, and the position for another photo come from the other parent. To do this, *parameter* had to be defined in such a way that an arbitrary number of free variables

could be grouped together to make one parameter. Also, it had to be possible to add and scale these parameters. All of this is the reason for the Parameter interface defined in the DE package. With this grouping of free variables, the optimization problem has only one parameter for each photo and maybe one extra parameter for the width variable described above.

As a result of the special treatment applied to the first two surface points and the first two photos, the position of the camera for the first two photos has to be parameterized in a way that differs from all the other photos. Since the position of the camera for the first photo is confined to a half plane and has to project the first two points exactly, it can be parameterized by a single degree of freedom: the angle of elevation above the plane $z=-0.5$. The camera's distance from the point *(0,0,−0.5)* is defined as:

$$radius=\frac{\sqrt{1+\cos^2(\vartheta)-2\sin^2(\phi)\cos^2(\vartheta)+2\cos(\phi)\cos(\vartheta)\sqrt{1-\sin^2(\phi)\cos^2(\vartheta)}}}{2\sin(\vartheta)} \tag{6}$$

where ϑ is the angle between the two points as observed in the photo and ϕ is the angle of elevation for the photo. Note that equation 6 assumes that the first two points are located at *(0,0,0)* and *(0,0,−1)*. The position of the camera for the second photo could have been parameterized as described earlier in this section, or it can be parameterized in a manner similar to the first photo. The latter option is the one I choose and in this case, the only difference between the first and second photos is that the second photo also has an angle of rotation about the z−axis describing its position. Equation 6 applies as is for the second photo, just as it does for the first.

Although it would be technically possible to parameterize all photos in which the first two points appear in the same way as the first two photos, I have issues with this idea. This parameterization of the first two photos essentially defines the error associated with the observations of the first two points to be zero. Since these observations are no more or less likely to be in error relative to any other observation, making this definition is not necessarily a good idea. Therefore, my desire is to limit the polar parameterization to the first two photos.

I had difficulty getting the optimization problem to converge when all of the photos were considered simultaneously from the start. To reduce the search space and increase the rate of convergence, I decided to add the photos in one by one. I start with the two photos that have the largest number of points in common, run it for some period of time to reduce error, then add in random locations for the camera of the next photo to the population and run the algorithm again for some period to reduce error. Determining how long to run each sub−problem for is a bit of a challenge. On the one hand, running the problem for a long period of time reduces the mean error in the population and can improve the time to convergence later; however, on the other hand, the longer the sub−problem is run for, the less diversity its population has and the greater the risk that the global minimum might be missed. To avoid the latter problem, if the diversity of the population for some sub−problem drops below a set threshold, then the sub−problem is stopped and the next photo is added.

To measure the diversity of a population, I compute the standard deviation of the position of the camera for each of the photos and divide them by the standard deviation of the position of

the camera across all of the photos. The mean of this ratio across all photos is taken to be the diversity of the population. The number is always in the range 0 to 1 which makes it very convenient to work with. If the diversity of the population appears to consistently drop too low, the problem can be rectified by either increasing the scale factor for the random mutations, decreasing the mean number of parameters taken from the mutated parent vector when creating a test vector, or by increasing the size of the population. All of these are parameters that the user can tweak to get more appropriate behaviour from the optimization algorithm.

For the final optimization problem with all of the photos included, the diversity measure can be used to help decide when the problem has converged to a solution. Left to run, the diversity will asymptotically approach zero as the problem converges; hopefully to the global minimum.

To help get a feel for how the optimization process is progressing, a simple 2 dimensional display of the camera and point positions was implemented. This view is a projection of the scene onto the x−y plane, and since the coordinate system is defined in a somewhat arbitrary manner (by the two points that happen to be decided to be the first two), it can be relatively difficult to interpret the results displayed in this 2D view. Regardless, they can be interpreted and do provide some insight into how the optimization is progressing.

When the optimization process is complete (and periodically throughout just in case it never finishes), the results of the optimization algorithm are written to two different output files. One contains the current state of the problem (represented by the DE object) in the format of a Java binary object stream. This file can be later used to resume an optimization; however, it is a rather delicate format and any changes to the source code and recompilation causes older versions of these binary files to become unreadable. The other file created by the optimization program is a plain text file containing all of the parameters for camera position, orientation, and point location. Here is sample of this output file:

```
1 0.0 0.0 0.0
13 0.0 0.0 −1.0
2 0.024 0.121 −0.078
3 −0.024 −0.007 −0.054
6 0.019 0.087 −0.188
  ...
16 −0.804 −2.773 1.644

6 −2.547 −0.0 −1.042 0.970 −0.065 0.230 −0.157 0.548 0.820
4 −0.673 2.237 0.774 0.280 −0.854 −0.436 0.545 −0.232 0.805
  ...
2 −0.813 −2.793 1.654 0.350 0.741 −0.571 −0.111 0.639 0.760

5.00E−4
error: 7.340
diversity: 0.001856
generations: 33000
```

For a better way to visualize and interpret the results, a Java3D program was written to display the results read in from either the binary stream or the text output. This program

displays the following items: all of the surface points as small coloured spheres with text labels; each of the photos as a cube to represent the camera, a small thumbnail version of the actual photo, projection lines to each of the points, and text labels if desired; and the 3D convex hull of the surface points as either a wire frame of a solid object.

The visualization program supports movement throughout the space via the cursor keys on the keyboard. Such movement can be done in one of two modes: polar or rectangular. Polar mode has at its center the mean of all the surface points and is generally easier to maneuver with; although, it doesn't always let the user view the scene from the most appropriate angle. In rectangular mode the user must orient their view in the direction they wish to move, and then press the key to go forward to that location. This is a little harder than it may sound but does allow one to view the scene from any desired angle.

Results

I tested the technique described above with three different subjects: one is a small stuffed penguin (the Linux mascot with the letters *sgi* on the front), another is a Logitech cordless trackball, and the third is the music room at the Graduate Residence here at the University of Toronto. Colour plates 1, 2 and 3 (included as separate files) show all of the photos analyzed during this project. In each photo, a small black cross has been superimposed to illustrate where each observation was made.

Table 1 summarizes the samples and some of the results that can be expressed numerically. For each subject, the size of the population used to do differential evolution was 400, the parameter *F* which controls the magnitude of the random mutations was set to *0.5*, the parameter *CR* which controls the cross−over function was set to *0.3*, and all sub−problems were run until the diversity dropped below *0.8* before the next photo was added. Each run of the optimization program required between *5,000* and *15,000* generations to converge to a solution. The mean error figure presented in table 1 can be considered to be measured in pixels per observation. For Tux and the trackball, the mean error is as low as could be expected considering the method used to input the observations into the computer. I suspect that the reason for the error being so much higher with the music room sample is a breakdown of my pin−hole camera model of the camera. I probably need a more sophisticated model of the camera when its wide−angle lens is attached to correctly handle the music room problem. The diversity of the final population is a number between 0 and 1 (with higher numbers meaning greater diversity) and can be thought of as a percentage if multiplied by *100*.

Subject	Photos	Surface Points	Observations	Facets in Convex Hull	Mean Error	Diversity of Final Population
Tux (penguin)	6	14	45	20	2.15	0.00020
Trackball	8	14	51	18	1.84	0.00018
Music Room	9	21	85	28	16.29	0.00028

Table 1: Summary of subjects photographed and an indication of the quality of the results obtained. Mean error is measured in pixels and the diversity is a ratio.

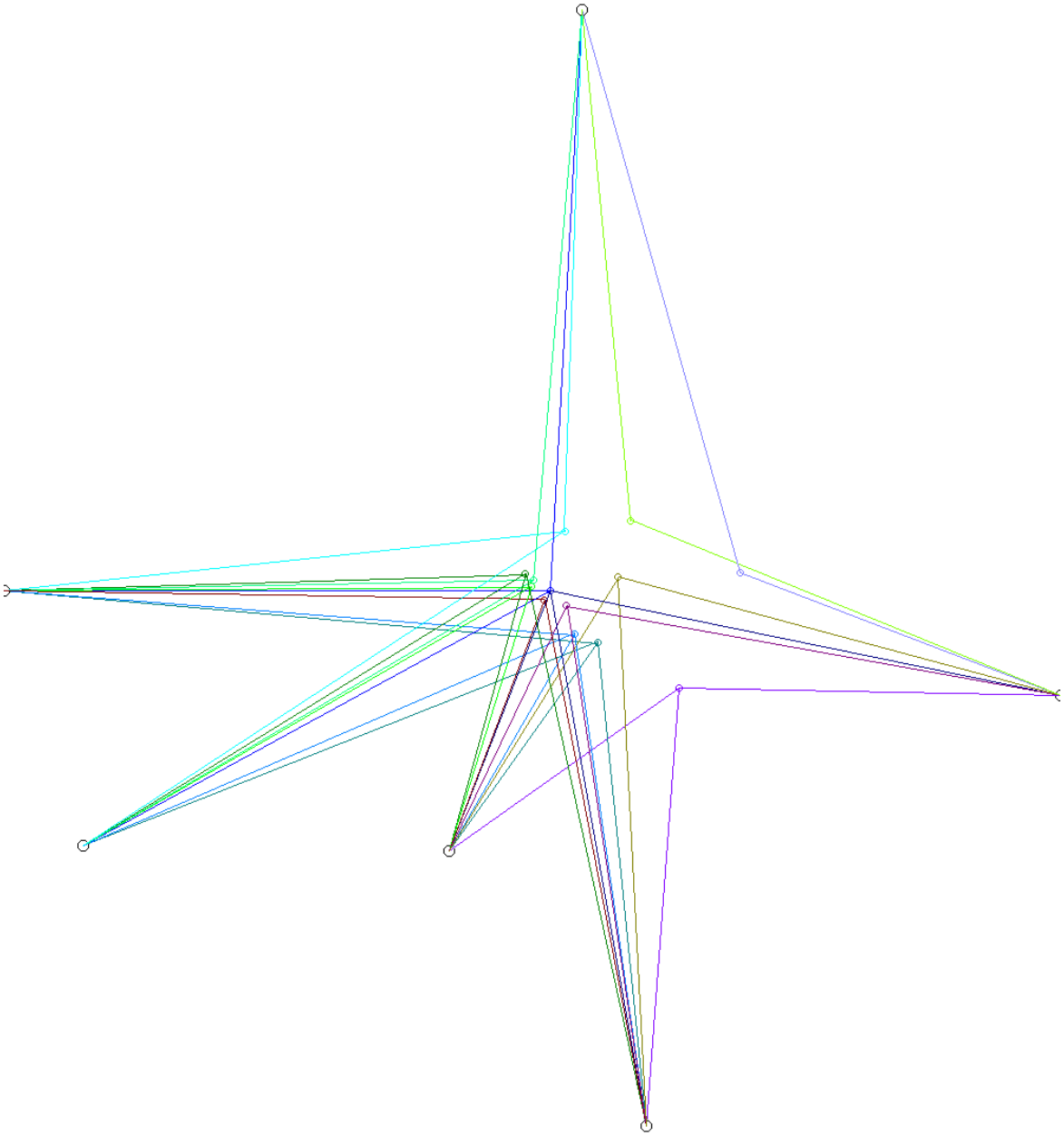

Figure 1: 2D view of the optimization of the penguin photos. The view is roughly from above with the penguin in the center. The large circles around the outside indicate the position of the camera when each of the 6 photos were taken. The smaller circles near the center represent points on the surface of the penguin.

As mentioned already, the 2D view generated by the optimization program is generally not very easy to read as it represents a projection of the scene onto some arbitrary plane. Regardless, figure 1 shows this 2D view for the case of the penguin sample. Here, the view is roughly from above. More precisely, the z−axis is a line running through the penguin's left eye and its right foot. The distribution of camera positions is pretty easy to see and one can even tell which photos were from the elevated locations. The 2D view associated with the other subjects are much harder to read and are therefore not presented here.

The best way to view the results with the Java3D viewer is in real−time with the program itself. Despite this, I do include two static views from the software here. Figure 2 shows a sample view of Tux. In this view it is quite easy to see all of the facial features of the penguin. Read the figure caption to find out which surface points are which.

I have not included any view of the trackball here because I didn't feel it would be instructive. These results can be viewed with the Java3D viewer program if necessary. Figure 3 shows a view of the music room from just outside the double doors. Although the results of this optimization problem are quantitatively quite poor, qualitatively, the location of the camera for each photo does appear to be very near the correct location and the features of the room do appear to be in roughly their correct locations.

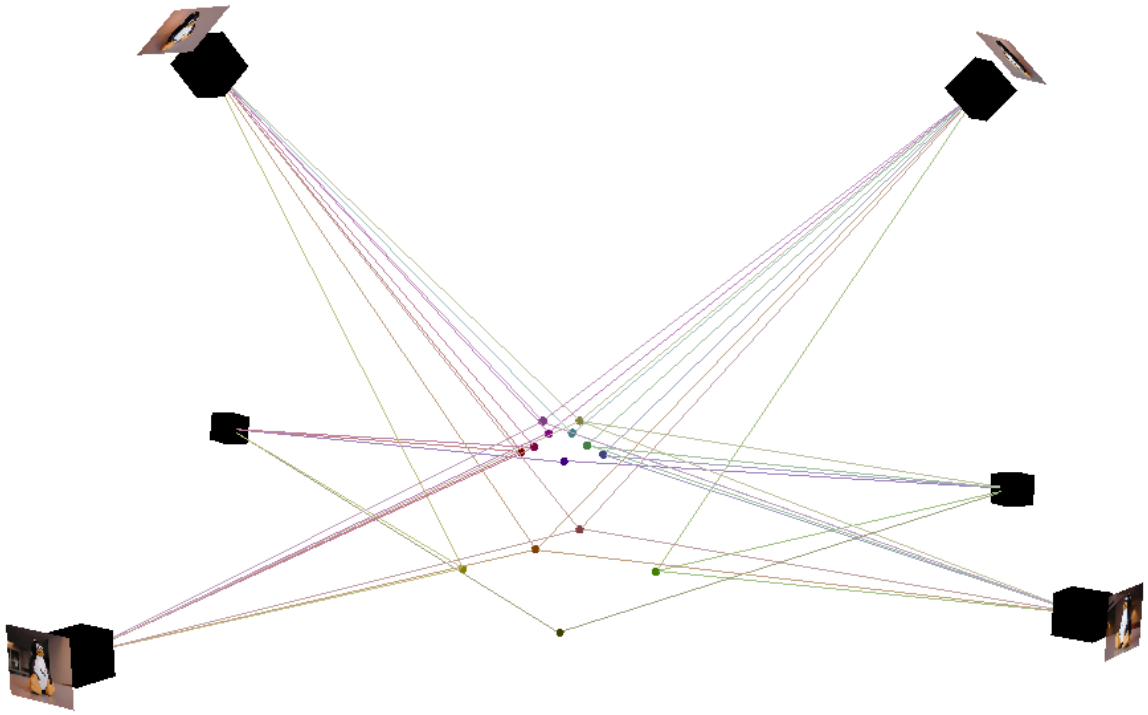

Figure 2: A sample of the derived penguin scene. The points on the surface of the penguin are, from top to bottom: the two eyes, the two nostrils, the two sides of the mouth, the two sides of the neck, a point on its back, the dot in the *i* of *sgi*, the bottom end of the *s* in *sgi*, the top of each foot, and, at the very bottom, the tip of the tail.

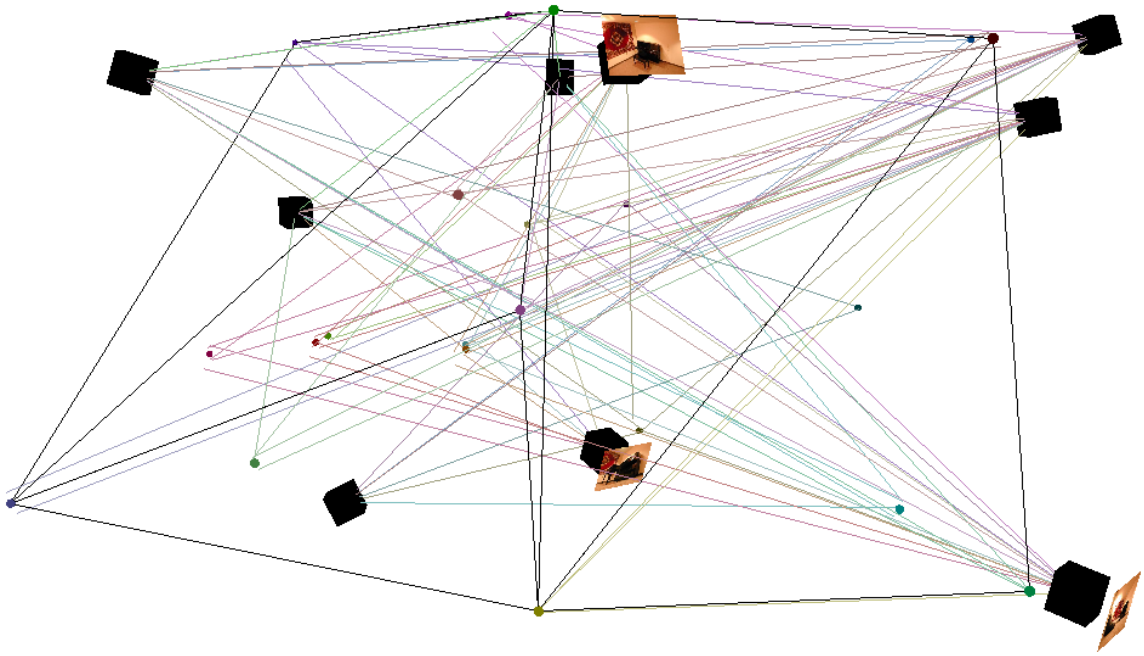

Figure 3: View of the music room scene from just outside the door. The front side of the convex hull is visible here and the outline of the double doors is clearly visible in the right half of the image. The point in the center of the image just to the left of the door is the light switch. The camera for the photos from the right appear outside the room, which is an error. The one camera on the left that appears outside the room is actually not outside as the top left corner of the room was not defined.

Conclusions

The problem of deriving the location and orientation of a camera when each of a series of photos was taken, along with a series of points on the surface of the subject of the photos is not a trivial one. It requires finding the solution to an overstated non−linear system of equations using an error function that defines a very complex surface with numerous local minima. Despite these difficulties, the technique of differential evolution appears to be effective for solving the problem.

For two out of three of the subjects I photographed, the global minimum of the error function appears to have been found and the error associated with the solution is as low as I could have expected. Despite my optimistic appraisal of the results, I do concede that without at least one sample taken with measured and calibrated camera positions, I have no way of knowing for sure whether my models of the scene match the reality in which the photos were taken.

The music room model failed to converge to a low error solution, but the lowest error solution that was found does appear to be somewhat consistent with the true scene. I suspect that my pin−hole model of the camera does not accurately apply in the case where a wide−angle lens has been used and the camera is set on its widest angle setting. If this is true, a better understanding of the optics of the camera should yield a much lower error solution.

I have several ideas as to how the technique presented here could be extended to provide a complete 3D model of the scene. First, if the location of the camera for each photo has been determined accurately enough, then Kutulakos and Seitz's space carving method [Kutulakos 1998] could probably be applied to find the maximum volume capable of explaining the photos (the photo hull as they define it). If the location of the camera cannot be determined with sufficient accuracy by my method, then Kutulakos's more recent work [Kutulakos 2000] in approximating an unknown scene from a series of photos could be applied instead.

Although Kutulakos space carving technique does work, I have some reservations regarding that method. Space carving works by starting with a volume that completely encompasses the subject of the photos and then repeatedly carving away individual units of volume until a volume that is consistent with the photos is found. Unfortunately, if one unit of volume too many gets carved away for some reason, the space carving method can be subject to catastrophic failure as the entire volume completely disappears. This potential for failure aside, I also feel that since my work here produces several points known to be on the surface of the subject, that a surface based approach to recovering the 3D shape of the scene would be more appropriate. With that in mind, I have two ideas.

If the topology of the subject can be assumed to be that of a sphere, then it may be possible to itteratively refine the convex hull of the known surface points. First, if there are any surface points that are not vertices of the convex hull, they have to be added to the hull. The projection lines that join each photo to the points that appear in that photo can be considered to be lines of sight. For each interior surface point, use these lines of sight to identify the facet of the convex hull that is most responsible for obscuring that interior point and then replace that facet with several new facets that include the obscured surface point as a vertex. Once all of the surface points are vertices of the closed (but no longer convex) hull, an itterative procedure whereby the largest facet of the hull (the one with the longest edge) is subdivided by adding a vertex at the center can be applied. After a facet has been subdivided, the vertex at the center needs to be moved up or down along the normal of the original facet to find a location for it on the surface of the subject. Some sort of center weighted error function could be defined to aid in this process. If this technique works, it would allow the mesh defining the hull to be refined to any arbitrary level of detail. This idea was inspired by the work of Hoppe, *et. al.* on optimizing meshes [Hoppe, 1993] and I believe that a variant of their mesh optimization algorithm could be applied to optimize the results of the technique described here to find an optimal mesh.

The other idea I have for recovering 3D shape was influenced by the work of Szeliski, *et al.* on modeling surfaces with dynamic particles [Szeliski, 1993] and, like their work, I believe this idea would be appropriate for recovering the shape of objects with arbitrary topology. The idea here is to first find a plane tangent to the unknown surface at each point on the surface that is known. The lines of sight mentioned in the preceding paragraph could be used to compute an initial estimate of the normal to the surface. Some sort of optimization could then be done to find a more accurate estimate of the tangent plane. Once a tangent plane is found, the estimate of the unknown surface can be extended by adding points within the tangent plane close to the points already known to lie on the surface. The surface can be extended in this way until the entire surface has been covered. At this point, Hoppe's technique (or a variation of it) could again be applied to derive an optimal mesh representing the unknown surface.

I have to admit that my original plan for this project was to try out the techniques and ideas just mentioned in the previous two paragraphs. Unfortunately, I was a little naive regarding the difficulty of deriving the location of the camera for a series of photos and that problem took far more time than I had hoped. Despite this, the work presented here was still very beneficial to me and I hope that I find time to continue to play with some of the other ideas and techniques discussed in this paper.

For source code and other materials related to this project, see:

http://www.cs.utoronto.ca/~cvs/geometry/

References

K. Kutulakos and S. Seitz. A theory of shape by space carving. *Technical Report TR692*, Computer Science Dept., U. Rochester, 1998.

K. Kutulakos. Approximate N−View Stereo. *Proceedings, 6th European Conference on Computer Vision*, pp.67−83, Dublin, Ireland, 2000.

M. Caceci and W. Cacheris. Fitting Curves to Data. *Byte Magazine*, May, pp. 340−362, 1984.

K. Price and R. Storn. Differential Evolution. *Dr. Dobb's Journal*, April, pp. 18−24, 1997.

H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh Optimization. *Computer Graphics (SIGGRAPH '93 Proceedings)*, pp. 19−26, 1993.

R. Szeliski, D. Tonnesen, and D. Terzopoulos. Modeling Surfaces of Arbitrary Topology with Dynamic Particles. *Proc. Computer Vision and Pattern Recognition Conference (CVPR '93)*, New York, NY, June, pp. 82−87, 1993.