A 3-D MODEL EXTRACTION SYSTEM

Robert L. Russell, Richard A. McClain, and B. Patrick Landell
GE Aerospace Advanced Technology Laboratories
Moorestown, NJ 08057

ABSTRACT

This paper describes PolyFit, a 3-D feature extraction system which allows a user to interactively extract three dimensional models from photographs with little apriori information. PolyFit's algorithm for simultaneously determining the camera parameters and scene geometry is a nonlinear least squares optimization. The computed geometry and camera information enables photographic texture extraction from the source imagery and subsequent rendering of the scene geometry from arbitrary view points. The PolyFit user interface provides tools which streamline the model building process as well as means for model inspection and exploitation. PolyFit has been shown to provide a 10 to 1 productivity improvement over previous manual methods.

Keywords: Computational geometry, 3D Feature Extraction, computer image generator (CIG), photo-texture extraction, camera modeling, object and scene modeling, mensuration.

1. INTRODUCTION

Training through computer image generation, systems today are challenged to provide the most photo-realistic renditions of real life environments [1]. The systems which provide high speed photo-realistic rendering require accurate models of the world accompanied by precisely registered photo-texture. The cost, however, of developing the databases required for this realism can be staggering. This high cost is a direct result of the time and manpower currently needed to generate a database of any significant scale [2,4]. This long database construction time also limits the use of simulation systems for applications such as mission planning or rehearsal because timely use of recent photo-reconnaissance imagery is not generally achievable [3].

One aspect of database development which is particularly tedious is the modeling of architecture within the gaming area. In the past this has been accomplished by manual photointerpretation techniques. Modelers would attempt to extract the geometry of a given building by trial and error using the available imagery as reference and inputting the computer description manually. Scale would be estimated from visual cues such as the height of a doorway or the length of a recognized vehicle. For buildings exhibiting simple geometry this technique worked well enough. However, as the complexity of the building increased, the accuracy of the model decreased. Imagine trying to extract the angle between two edges (other than 90 degrees) in a 2-D image taken from an oblique perspective. Furthermore, placing the building in the database would require more trial and error by the modeler to determine the relative location of this building with respect to its neighboring buildings.

A secondary problem with previous manual approaches is that the resulting models are not registered to the imagery. Thus, to extract photographic texture from the image, the computed object wireframe would be interactively manipulated to approximate the orientation, position and scale of the object within the image; a very time consuming process.

Previous approaches to speeding this object modeling process have often made the assumption that the available imagery already has associated camera model information registered to terrain elevation data. These approaches then attack the problem by letting the operator place an object in the world by manipulating a wireframe over the image of the object. The camera model is used to determine the object's scale, though A. Hanson et. al [2] also used solar illumination geometry to better determine object height for near nadir views. These previous approaches are limited because: 1) they can only handle simple geometries, 2) they rely on the existence of supplementary data and 3) they don't optimally fuse the information from multiple images in a single 'best fit' of the object. This fusion aspect will be further explained in later sections.

2. POLYFIT OVERVIEW

The system described in this paper has been designed to overcome the above mentioned limitations. This system, referred to as PolyFit, extracts complex 3-D models from single or multiple photos with little apriori information. Image camera models, if not provided, are computed along with the object models. If maps or control points for the images are available PolyFit can locate the models precisely in the world, otherwise, the database is defined relative to a user definable local coordinate system. Furthermore, PolyFit achieves high accuracy by fusing the information from all sources into one best fit solution. The PolyFit solver uses a constraint elimination procedure and the Gauss-Newton algorithm to solve the constrained nonlinear optimization. Upon solution the 3-D models are registered within the imagery allowing the photo-texture to be easily extracted and orthorectified for convenient access by the CIG. Using PolyFit's own rendering capability allows: 1) verification of geometry, 2) inspection of photo-texture and 3) examination of the model from arbitrary vantage points.

Based on results received from GE's Simulation and Control Systems Department (SCSD), which builds databases for their COMPU-SCENE line of CIGs, PolyFit provides a 10 to 1 speed improvement over previous manual methods. The database for which SCSD employed PolyFit was later used to demonstrate the performance of their COMPU-SCENE VI CIG at the Interservice/Industry Training Systems Conference held in Orlando, Florida in early November of 1990.

In addition to using the database as input to a CIG, further exploitation of the data can be achieved through PolyFit's mensuration capabilities. This facility allows the operator to query the database interactively to extract geometric information useful for applications such as mission planning. For example, one could request the distance between buildings or the height of a window from the ground.

A future enhancement will perform reasoning on the data to compute, for example, the least detectable path in moving from point A to point B, given the location of likely sentry positions.

3.0 THE SOLVER

The essence of PolyFit is to rely on the human to surmise the general layout of the scene and to designate within the images the locations of the scene features, functions for which a human is extremely adept, while the computer solves a least square error optimization to recover the scene geometry with high accuracy, a function best performed by the computer. This section describes the latter component, the PolyFit solver, which simultaneously computes the camera parameters and scene geometry based on the principle of maximum likelihood, treating the measurements of the human user as noisy observations.

The inputs to the solver are the scene topology, scene geometric constraints, and for one or more images, a list of designated vertex positions in the images. The positions and focal lengths of the cameras are generally unknown. The problem is to recover the scene geometry from the positions of features in the images.

3.1 Scene Model

The scene model consists of some number of objects. The position of each object is represented by a coordinate frame consisting of a rotation matrix and translation vector. Objects themselves are modeled as a constrained polygonal mesh, consisting of vertices, lines, and planar faces. The polygonal mesh represents the visible side of object surfaces. Each object consists of a number of vertices: $\mathbf{v}_i$, $i = 1,\ldots n$ (3-D vectors). Each face of an object is defined by a sequence of vertices clockwise around the face when viewed from the visible side. Lines connect adjacent vertices on the boundary of a face. By introducing another geometric entity, direction vectors, constraints can be placed on line directions or face normals.

The information describing the relationships of all the geometric entities is stored in a **scene structure graph**. The structure graph consists of a list of objects; objects contain lists of vertices, faces, and lines; faces contain lists of vertices and lines; etc. The structure graph is the topological description of the scene.

The parameters which instantiate the topology into completely specified models are the **scene parameters**. Each geometric entity in the model is defined by some number of parameters, which may, for convenience, exceed the minimum number of parameters or degrees of freedom (DOF) needed to specify the entity. Vertices are defined by a 3 vector, $\mathbf{v}$. Lines are defined by a line direction vector $\mathbf{a}$ and a vector offset $\mathbf{p}$. Points satisfying $\mathbf{x} = k\mathbf{a} + \mathbf{p}$ for some scalar k lie on the line. Faces are defined by a face normal $\mathbf{a}$ and any point on the plane $\mathbf{p}$. Points satisfying $\mathbf{a} \cdot (\mathbf{x} - \mathbf{p}) = 0$ lie on the plane. Direction vectors are represented by vectors. Determination of the scene variables (all the parameters together) is the crux of the reconstruction problem.

3.2 Scene Geometric Constraints

Some constraints arise implicitly from the definition of object models as a planar mesh because all vertices in a face must lie in the plane of the face. This constraint is expressed as $\mathbf{a} \cdot (\mathbf{v} - \mathbf{p}) = 0$ for each vertex in a face.

Additional explicit constraints on the scene geometry are needed when the camera model and scene geometry are underspecified. For example, constraints may be used to fill in parts of a scene not viewed in any image. Also the user will often desire to force scene models to satisfy particular conditions, such as walls being vertical. The current system provides four types of constraints: 1) point constraint - the user directly specifies the coordinates of a vertex, 2) direction constraint - line directions or face normals are constrained to lie along a particular direction, which can be fixed or free to vary, 3) length constraint - the length of a line is fixed, and 4) coincidence constraints - a vertex, line, or face is constrained to be coincident with another vertex, line, or face.

3.3 Constraint Elimination

Our approach to handling scene geometric constraints has been to eliminate them while simultaneously reducing the number of scene variables. To accomplish this, the variables are grouped into subsets, called elements, associated with the basic geometric entities of the model: coordinate frames, vertex points, lines, face planes, and direction vectors. The constraints (point, direction, length, coincidence) each define a relationship between two elements. As an example, if a face contains a vertex, then the face plane and the vertex point must intersect. To eliminate this coincidence constraint, the symmetry is broken and one of the elements is placed as a constraint on the other. Either the face is free and the vertex must lie in the plane of the face, or the vertex is free and the face must intersect the vertex. The constrained element has reduced degrees of freedom (fewer variables) due to the constraint. Continuing the example, if the face is constrained by the vertex, then the face is no longer represented by a free plane with 3 DOF but by a plane with 2 DOF which is attached to a fixed point in space. In effect, the

constraint disappears from the overall system and is submerged into the constrained geometric element. In implementing the system, 41 types of geometric elements were necessary.

Eliminating all constraints requires finding an acyclic graph, called the **dependency graph**, which defines an evaluation ordering of all elements. The ordering is such that if one element is dependent on other elements, then it must appear later in the evaluation ordering than all of the elements it depends on. In addition, individual geometric elements must be neither over- or under-constrained. For example, a plane cannot be constrained by four points, and 0 DOF must be allotted to a vertex which is not seen in any camera view.

The solver finds a legal dependency graph with a two phase search procedure. The first phase is a top down search that establishes basic ordering information by determining the minimum depth in the dependency graph of each element, while ignoring overconstraining elements. The second phase is bottom up search that generates the final ordering. The trial selection of elements is based on the depth number computed in the first phase. During the bottom up search, as each element is placed into the evaluation ordering, checks are made that all constraints are compatible and that no element is either over- or under- constrained. The search procedure will resort to an exhaustive search with backtracking to find a compatible ordering, but before this begins, the graph is normalized to localize the search space. The exponential runtime associated with a backtracking search is not a problem due to all the preparations that precede it that effectively guarantee that the search time will be very brief.

3.4 Solution Formulation

Let $\mathbf{x}$ be the result of concatenating all the scene free variables into a single vector, of dimension N = the number of degrees of freedom of the scene. The positions of the vertices can be extracted from the scene parameter vector, hence, let the position of the i'th vertex be denoted $\mathbf{v}_i(\mathbf{x})$. The remaining variables in the system are the unknown camera parameters, denoted by $\mathbf{c}_j$ for the j'th of m cameras. The predicted image position of the i'th vertex in the j'th image is obtained from the camera parameters by applying a 3D to 2D camera projection function, $\mathbf{p}(\mathbf{v}_i, \mathbf{c}_j)$. The form of this function will depend on the type of camera used (frame, panoramic, etc). The current implementation assumes a frame camera.

For some combinations of vertex and image, an observation of the vertex in the image will be available. The observed 2D position of vertex i in image j will be denoted, $\mathbf{i}_{ij}^{o}$, and treated as a noisy observation with standard deviation σ_{ij} in both the horizontal and vertical directions. The set of ij pairs for which an observation is available will be denoted by the set D. For some vertices, a direct observation of its 3D position may be available, generally from a map or blueprint. The observed 3D position of the i'th vertex will be denoted $\mathbf{v}_i^{o}$ having standard deviation σ_i in each of x, y, and z. The set of i's for which 3D control points are available will be denoted by the set C.

Finding the scene variables and camera parameters corresponding to the most likely set of noise samples (maximum likelihood estimate) is readily shown to be equivalent to the following optimization:

$$\text{Minimize } f(\mathbf{x},\mathbf{c}_1,\ldots,\mathbf{c}_m) = \sum_{ij\in D}\left|\frac{\mathbf{i}_{ij}^{o}-\mathbf{p}(\mathbf{v}_i(\mathbf{x}),\mathbf{c}_j)}{\sigma_{ij}}\right|^2 + \sum_{i\in C}\left|\frac{\mathbf{v}_i^{o}-\mathbf{v}_i^{o}(\mathbf{x})}{\sigma_i}\right|^2 \quad (1)$$

This optimization is unconstrained, and can be restated in more general terms by concatenating all unknowns into one vector, $\mathbf{z}$; concatenating all modeled observations into one vector function, $\mathbf{g}(\mathbf{z})$; concatenating all observation measurements into one vector, $\mathbf{g}^{o}$; and combining the standard deviations of observation errors into a diagonal weight matrix, $\mathbf{W} = \text{diag}(1/\sigma_i)$. Then, the optimization becomes:

$$\text{Minimize } f(\mathbf{z}) = \sum\left(\frac{g_i(\mathbf{z}) - g^{o}}{s_i}\right)^2 = \left|\mathbf{w}(\mathbf{g}(\mathbf{z}) - \mathbf{g}^{o})\right|^2. \quad (2)$$

3.5 Solving the Optimization

Initial Approximation. Solving the optimization with the Gauss-Newton procedure requires an initial estimate reasonably close to the solution. The solver has a procedure to bootstrap from whatever apriori information is available (constraints plus measured vertex positions) to obtain initial estimates of unknown camera and scene parameters. Algorithms were developed which could use any available control points, or direction constraints, or simply corresponding vertices between cameras to generate initial estimates that are usually accurate enough to allow convergence of the subsequent iteration.

Summary of initialization procedure

1. Unless already known, assume cameras are located infinitely far from the scene.
2. Initialize orientation, translation, and scale of cameras (if possible) using any "knowns" visible to the cameras:
   - lines in known directions
   - control points
   - approximated vertex locations from step 3.
3. Use stereo to approximate vertex locations (ignore constraints).
4. If some cameras are still unknown, go back to step 2.
5. Call each elements initialization procedure in an order such that all elements it depends on will be called first (always possible for an

acyclic graph, which can be topologically sorted).

Iterative Improvement to Final Solution. Solving (2) is the same as finding the least squared error solution to

$$\mathbf{W}\mathbf{g}(\mathbf{z}) = \mathbf{W}\mathbf{g}^{o}.$$

These nonlinear equations are solved, in a least square error sense, using Gauss-Newton's method, which iteratively improves an approximate solution $\mathbf{z}_n$. Each iteration consists of:

Step 1: Compute Jacobian matrix, $\mathbf{g}'(\mathbf{z}_n) = \partial\mathbf{g}/\partial\mathbf{z}$, which provides a linearized system of equations, valid in the neighborhood of $\mathbf{z}_n$:

$$\mathbf{W}\mathbf{g}(\mathbf{z}_{n+1}) = \mathbf{W}\mathbf{g}(\mathbf{z}_n) + \mathbf{W}\mathbf{g}'(\mathbf{z}_n)\,\mathbf{d}_n = \mathbf{W}\mathbf{g}^{o}.$$

Step 2: Solve the linearized system for the least squared error differential change, using the pseudoinverse*

$$\mathbf{d}_n = -(\mathbf{W}\mathbf{g}'(\mathbf{z}_n))^{\dagger}\mathbf{W}\,(\mathbf{g}(\mathbf{z}_n) - \mathbf{g}^{o}).$$

Step 3: Apply the differential change to $\mathbf{z}_n$:

$$\mathbf{z}_{n+1} = \mathbf{z}_n + \mathbf{d}_n.$$

Gauss-Newton's method converges to a local minimum mean squares error solution of $\mathbf{W}\mathbf{g}(\mathbf{z}) = \mathbf{W}\mathbf{g}^{o}$. The procedure described in the previous section provides an accurate enough starting point to allow convergence in most cases.

For the times when the initial guess is not close enough (and convergence is not obtained), an extra solve mode has been implemented which essentially ignores model constraints (excepting those needed to establish a base reference coordinate system) and computes camera models by allowing the 3D vertices to be unconstrained. A subsequent overall solve has converged in all cases where a solution has been well defined.

Computation of Jacobian. Due to the chaining of elements in the dependency graph, a single free parameter can affect the positions of many vertices. In order to develop an automatic procedure for computing the Jacobian matrix, a small chain rule procedure has been implemented for each of the 41 geometric elements, which computes the partial derivative of the element in terms of the partials of all the terms it depends on.

* The pseudoinverse is computed according to: $A^{\dagger} = (A^T Z)^{-1} A^T$. In the actual implementation, the sparsity of the Jacobian matrix has been exploited to provide a fast response.

With these chain rule procedures in place for all geometric elements, the computation of the Jacobian matrix proceeds as follows:

for each free parameter λ
  let $\mathbf{q}$ = the geometric element associated with λ
  initialize $\partial\mathbf{q}/\partial\lambda$ (to the appropriate unit vector)
  for every element $\mathbf{r}$ affected by $\mathbf{q}$
    compute $\partial\mathbf{r}/\partial\lambda$ using the elements chain rule procedure

In this procedure, in the loop where the partials of all affected elements are computed, the partial derivative functions are called in dependency order.

4.0 THE USER INTERFACE

4.1 Design Goals

The interface was designed with the assumption that the user has a working knowledge and understanding of the scene model described in section 3.1. As the user builds the scene model, textual and graphic feedback is provided to indicate each additional feature. Two windows provide different types of textual feedback; a left window (referred to as the message window) handles prompts and error messages while a right window (referred to as the model listing window) gives a textual description of the complete scene model. The model listing window is constantly updated as the user makes changes to the scene model. Graphic feedback is provided by a bottom window of the interface, referred to as the canvas. The source imagery is also displayed on the canvas. As new features are added to the scene, model overlays are provided here to represent them. Since displaying all features simultaneously would cause the canvas to be very busy, certain features, such as constraints, are only displayed upon user request.

The challenge of designing PolyFit's user interface was to provide a convenient direct manipulation what-you-see-is-what-you-get (WYSIWYG) environment. The goal of the WYSIWYG environment is to give the user a clear indication of the current state of the scene model in order to minimize user memory load, prevent errors, and allow for easy mental pacing through the steps of building a model. The model listing window, for example, has been found to be very effective at reducing user memory load by providing a quick way to survey an object's complete state. This, combined with the graphic display, also helps the operator to readily spot mistakes rather than exhaustively search for them. The feedback also provides reassurance that user requests have been detected and correctly interpreted. Finally, the graphic cues can be exploited as references for faster user inputs. These, as well as other usability heuristics, are more formally presented in [5]. Other examples of PolyFit's user feedback are described in ensuing paragraphs.

In order to reduce training time and push PolyFit toward a walk-up-and-use system the 'Design-for-Successful-Guessing' strategy as presented in [6] is employed. This strategy includes design principles such as: 1) make the repertory of available actions salient, 2) offer few alternatives and 3) require as few choices as possible. The effects of this strategy mostly concern PolyFit's main panel. The main panel is the window located below the message and model listing windows described above and contains menus, buttons, toggle switches and sliders. The menus reduce the user's alternatives by grouping together sets of related choices. The buttons most frequently used are located near the bottom, nearest to the canvas where the images are displayed. Buttons concerning similar actions are grouped together and the groups are ordered in a hierarchy indicating the basic order of actions needed to build a model. The label for each button is short, but descriptive, and the exit is clearly marked.

4.2 Model Building

The modeler's major task in generating the scene model is describing the object topologies. This may require first analyzing the imagery to develop a mental picture of each object's shape. To aid the modeler, PolyFit provides a convenient means for quickly surveying the available imagery, thereby saving the user from having to remember which images contain which views. A palette can be brought up which displays a small subsampled copy of each image, allowing the user to select the pertinent views. Image contrasting facilities are also provided to improve image quality through interactive adjustment of image brightness.

The user begins to build the object topology by designating the location of object vertices in the images with a mouse. Each time a new vertex is designated another vertex is added to the object's topology and an image icon is generated to show the location of the designation. In order to achieve subpixel accuracy on the vertex designations, a local zoom can be performed on the area around the vertex. If a vertex is visible in multiple images the user can designate new vertex views for each image. Conversely, if a vertex is occluded in one or all images it can still be designated, but, the image location of the designation will only be used to display an icon, which then can be used to interactively access the vertex.

The object's faces are created by selecting the vertices which make up the face, using the mouse and the previously designated icons. Vertex selection is accomplished by clicking the mouse near a vertex icon. Traversing the face in an outward viewing clockwise order defines the face with an outward pointing normal. Lines are automatically created for each edge of the face. The face is textually described in the model listing window by listing its vertices by number. The canvas icons for each vertex created during the vertex designations mentioned above can be converted to their respective numerical value via a menu option for comparison with the model listing.

Once the object topologies are defined, the user can then assert constraints on the object geometries. The types of constraints available to the user were described in section 3. The procedure is to select the type of constraint from a menu and then select the geometric element to be constrained by selecting vertex icons within the image display area.

The next logical step in the process is to invoke the solver. The solver first determines whether the problem set as currently defined by the user's inputs is solvable. This determination is made during the solver's constraint elimination process. The most common condition that causes an unsolvable problem set is that the system is underspecified. In this situation PolyFit informs the user which vertex, line, or face is underspecified. The user then has the option of designating more vertices or providing additional constraints. Another condition that can cause an unsolvable problem set is when incompatible constraints have been defined by the user. In this situation PolyFit identifies the overconstrained element, and the user simply deletes incorrect constraints. If the problem set is solvable, then the system computes the best fit camera/model geometry according to the procedure described in section 3.

After the geometry of the model has been solved it is important to have a means of verifying its integrity. PolyFit on request will overlay a wireframe plot of the model on the images used to construct it. The wireframe plot can also be interactively manipulated to inspect the back side of buildings or check for missing faces. When the solver is invoked the overlay is automatically displayed at each solver iteration to show the progress of the computations.

4.3 Library Objects

While PolyFit as described thus far significantly speeds the construction of architectural models, other features of the interface provide further productivity improvements. These enhancements concern the use of library objects. Library objects are designed to relieve the operator of constructing much of the topological and constraint information associated with a given building. Instead of constructing every building from scratch, common or previously modeled shapes can be selected from a palette of wireframes. Selection of a library object allows the system to automatically create all object vertices, faces and internal constraints. The operator is then left with the task of vertex view designations.

View designations are accomplished by associating a wireframe vertex with an image location. The wireframe can be translated and rotated in order to aid the operator in making the correct association of wireframe vertex to object vertex. Not all vertices of the wireframe have to be designated; only enough to allow the undesignated vertices to be describable in terms of the designated vertices and the constraints. The effect is to provide the operator with a simpler, more intuitive interface, thus reducing training time and increasing productivity.

4.4 Photo-Texture Extraction

Photo-textured rendering requires the creation of texture maps to be applied to each object face. PolyFit allows an operator to choose which image to extract the photo-texture from for each face. If a face

is not visible, a texture or color for the face can be chosen elsewhere.

CONCLUSION

We have presented an interactive technique for extracting 3-D models from single or multiple photographs. This system does not require precomputed camera models, terrain elevation data or ground control points. An algorithm was discussed which fuses all input data into an overall best fit for the model. PolyFit outputs polygonal mesh polyhedral models and, if not provided, camera models. PolyFit also extracts photo-texture for each face of the polygonal mesh for realistic rendering from new vantage points. The model construction process has been streamlined by a sound graphical user interface.

PolyFit's success comes from its ability to combine the human's photointerpretation skills with the computer's computational capabilities. Success has been demonstrated through extensive use by modelling engineers of GE's Simulation and Control Systems Department.

REFERENCES

1. Economy, R., Bunker, M., "Advanced Video Object Simulation," Proceedings of the IEEE 1984 National Aerospace and Electronics Conference, (May 1984), vol. 2, pp. 1065-1071.

2. Hanson, A., Pentland, A. and Quam, L., "Design of a Prototype Interactive Cartographic Display and Analysis Environment," Proceedings of Image Understanding Workshop, (February 1987), pp. 475-477.

3. Donovan, K, "Mission Rehearsal Database Requirements and Technologies", Proceedings of the Interservice/Industry Training Systems Conference, (November 1990), pp 157-162.

4. Biesel, H, "From Source Materials To Data Bases: Higher Fidelity at Lower Cost", Proceedings of the Interservice/Industry Training Systems Conference, (November 1990), pp 163-171.

5. Nielsen, J., Molich, R., "Heuristic Evaluation of User Interfaces," Proceedings of Association for Computing Machinery's Special Interest Group on Computer Human Interaction, (April 1990), pp. 249-256.

6. Lewis, C., Polson, P., Wharton, C. and Rieman, J., "Testing a Walkthrough Methodology for Theory-Based Design of Walk-Up-and-Use Interfaces," Proceedings of Association for Computing Machinery's Special Interest Group on Computer Human Interaction, (April 1990), pp. 235-242.