Directed Acyclic Graph Representation of Deformable Models

Siome Goldenstein[1] Christian Vogler[1] Dimitris Metaxas[2]

[1]CIS Department - University of Pennsylvania
200S 33rd Street, Philadelphia, PA 19104
[2]Department of Computer Science - Rutgers University
110 Frelinghuysen Road, Piscataway, NJ 08854
[1]{siome,cvogler}@graphics.cis.upenn.edu [2]dnm@cs.rutgers.edu

Abstract

Deformable models are a useful tool in computer vision and computer graphics. A deformable model is a curve (in two dimensions) or a surface (in three dimensions), whose shape, position, and orientation are controlled through a set of parameters. Deformable models can represent manufactured objects, human faces and skeletons, and even bodies of fluid.

In computer graphics we use deformable models for animations and simulations, whereas in computer vision applications, such as tracking and fitting, deformable models help to restrict the family of possible solutions.

*In this paper, we introduce the use of a directed acyclic graph (*DAG*) to describe the position and Jacobian of each point on the surface of deformable models. This data structure, combined with a topological description of the points, is simple, powerful, and extremely useful for both computer vision and computer graphics applications. We show a computer vision application, 3D deformable face tracking, and a computer graphics application, cyberglove data visualization and calibration.*

1. Introduction

For years engineers and computer scientists have been dealing with different abstractions to represent real solid objects inside a computer. Once the computer has an internal model of an object, we can analyze the object's properties and simulate its performance (with finite elements, for example), we can transform the object to achieve special effects, and we can even try to recognize the object and its actions. Whenever the object's shape can change, according to a set of parameters, we can call it a *deformable model.*

Deformable models can represent a wide variety of things, from manufactured parts to the soft surface of the human body. It is a challenge to have a unified way to store and manipulate all these models. Also, the parameters can have different types of effects over a model. Usually, we classify the parameters in three types: those that do not change for a particular object (*static parameters*), those that change and affect all the model globally (*global parameters*), and those that change and affect only some regions of the model (*local parameters*). For example, in the model of a human face for tracking, the parameters responsible for deforming a generic face to a particular individual are the static parameters, the rotation and translation that place the face in the world are global parameters, and the parameters that control the raising of eyebrows or opening of the jaw are among the local parameters.

A deformable model based tracking application has a limitation: it only tracks objects that can be described by the chosen model. On the other hand, this restriction simplifies the problem and allows the algorithm to concentrate its attention to the change of the important information. Going back to the face example, it is more important to capture the opening of the mouth correctly than the individual 2D motion of all the image points of the mouth. Parameterizing the class of motion is even more important when we aim at 3D tracking from monocular images.

In this paper, we introduce the use of a directed acyclic graph (DAG) to represent a deformable model. This data structure treats all types of parameters consistently. It is easily extensible through the implementation of new building blocks, and allows an efficient integration between computer vision and computer graphics applications. Moreover, it creates a consistent way to describe deformations of a solid model that the computer can manipulate. As an implementation technique, it takes advantage of all the theory behind deformable models. The two most compelling advantages of this approach are that it becomes extremely easy to swap out one deformable model for another one without having to change the application code, and that the application framework can optimize the computations on the model dynamically, such as through elimination of common subexpressions.

2. Related Work

Snakes are one of the first examples of deformable models to appear in the literature of computer vision [13]. This theory was expanded to allow statistical representation of the shape of the models and its points, as well as an increase the number of applications [5, 3]. In our tracking example we use a general 3D deformable model framework [14] that has also been used for the estimation of shape from shading [19], for the combination of optical flow and edges in face tracking [7], and for the modeling of the human heart [15]. There are several studies on how to represent the static shape of family of objects, like a PCA decomposition [4], anthropometric characteristics [8, 9], or reconstruction from images [16] to describe the rest shape of human faces. Finally, there has been some recent and promising advances on how to dynamically improve and refine the shape and motion of a deformable model [20].

A fairly broad overview of the different methods for representation of objects in computer graphics can be found in [12]. We should point to the reader that one of the first constructive representations of objects, CSG (Constructive Solid Geometry), is a tree with boolean operations to construct the volume of the solid object [18]. Hierarchical deformations have been long used to represent skeletons and articulated rigid bodies in both computer graphics and robotics [6, 10], this hierarchy can be represented as a tree.

3. Deformable Models

The shape, position, and orientation of the surface of the model can change. These changes are controlled by a set of parameters $\vec{q}$. For every point i on the surface of the model, there is a function F_i that takes the deformation parameters and finds

$$p_i = F_i(\vec{q}),$$

where p_i is the point position in the world frame.

Computer vision applications need more than just the position of every point of the model for the given value of $\vec{q}$. In order to track or fit these parameters, most algorithms also require the knowledge of the first order derivatives. Since the specification of the deformations is part of the creation, or modeling process, of the deformable model, we assume that F_i, $\forall i$, is well-behaved: that is, that it has well-defined first order derivatives with regard to $\vec{q}$. The derivative of p_i with regard to $\vec{q}$ is the *Jacobian* J_i , where

$$J_i = \begin{bmatrix} | & & | \\ \frac{\partial p_i}{\partial q_1} & \cdots & \frac{\partial p_i}{\partial q_n} \\ | & & | \end{bmatrix}.$$

Each column l of the Jacobian J_i is the gradient of p_i with regard to parameter q_l.

In this section we describe a *graphical model*[1] for the general description of arbitrary deformable models. We represent a discrete model — only a finite number of points on the surface are represented —, and we use a graph to describe the functions that will generate the positions of the nodes. Although the model representation is discrete, points on the surface can be easily added, removed, and modified. Hence, it is straightforward to apply multi-resolution techniques with this data structure. We explicitly distinguish the topology of the points (how they are connected) from their geometry (where they lie in space), and describe them separately.

3.1. Topology Description

Deformations over the model affect only the point positions, so keeping the topology apart enables us to have different types of deformations applied over the same topological structure. Because of this dual representation, different models that share the same topology representation will have a bijective relation between their points.

In principle, the topology representation is unnecessary when all you need is a cloud of points. In practice, we are representing the surface of a solid model, and we need the connectivity to determine the visible points of the model, so the computer vision algorithms can know which nodes to affect (only the visible ones), and so that the computer graphics applications can correctly render the object. The topology description specifies the number of nodes (points) on the surface of the model, and how they connect. Figure 1 shows the topology of a simple cube, with one point on each corner.

3.2. Directed Acyclic Graph

The description of the model's geometry is more complex. While the topology usually remains static, the position of the points is controlled by $\vec{q}$, the parameter vector. We construct a *directed acyclic graph* (DAG) to represent the geometry. This data structure is capable of evaluating the position and Jacobian of all nodes given any value of the parameter vector $\vec{q}$.

[1] The term graphical refers to graphs, as opposed to computer graphics.

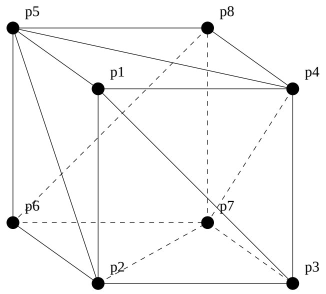

Figure 1. Typical topology of a cube. The eight corners are the only points represented in this model.

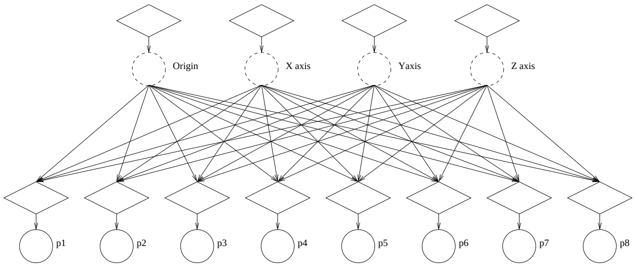

Figure 2. A simple DAG representation of the cube of Figure 3 parameterized by width, depth and height. Nodes are represented as circles, where the dashed ones are the ghost nodes. Dependencies are represented as diamonds.

This structure is composed of two basic elements: *nodes* and *dependencies*. Nodes can be *normal nodes* and *ghost nodes*. Normal nodes are the actual nodes on the surface of the model, and have their connectivity determined by the topology description. Ghost nodes, on the other hand, are not physical points of the surface. Ghost nodes can be seen as auxiliary points, temporary results or vectors. The dependencies are the mathematical building blocks that calculate the positions and derivatives in a recursive way.

Each node has one parent: a dependency, and each dependency can have multiple nodes as parents. The DAG representation of a mode is not unique. As an example, consider a parameterization of a cube, with the topology of Figure 1, according to its depth, height, and width.

A simple parameterization can be achieved with only one layer of dependency, illustrated on Figure 2 (where the nodes represented as circles, and the dependencies as diamonds). In this case, each corner of the cube is the sum of the origin and weighted vectors along the X, Y and Z axis.

Another possible DAG realization creates a ghost node to represent the center of the front square by subtracting half the depth along Z from the origin. The four corners of the front square come straight out of it, by adding and subtracting half of the height and width. From each of the the four frontal nodes we can just add the depth along Z to find the back corners. This complicated realization is illustrated in Figure 3, where dependencies are represented by diamonds, ghost nodes by dashed circles, and normal nodes are solid circles.

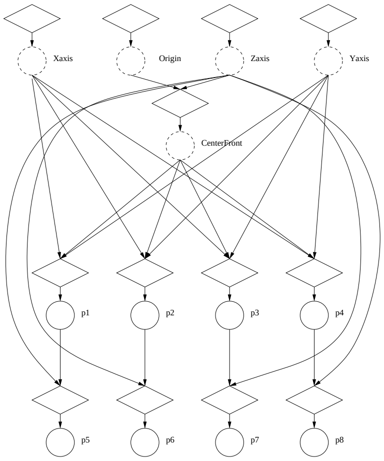

Figure 3. An illustration of a “not so simple” DAG representation of cube parameterized by width, depth and height. For a given model the representation in a DAG is not unique. Nodes are represented as circles, where the dashed ones are the ghost nodes. Dependencies are represented as diamonds.

Each dependency represents a basic operation, and has encoded knowledge on how to calculate positions and Jacobian, given a set of input nodes and parameters. Whenever we query a node for its position, or Jacobian, the node forwards the request to the respective parent dependency. The dependency in turn gets the position, or Jacobian, of the parent nodes, and calculates the resulting position, or Jacobian (using the chain rule). This structure defines a directed graph, and in order to avoid undetermined expressions[2], there are no loops. Thus, it is a *directed acyclic graph*. We evaluate all the information of the deformable model by evaluating every node. When the data structure is implemented with care, nodes and dependencies cache their values and there is no need to perform a topological sort beforehand.

[2] This is not entirely true, there are recursive mathematical expressions that are well defined, but to allow this class of expression would increase the complexity of the data structure and computational feasibility.

3.3. Dependencies

The dependencies are the mathematical building blocks of the DAG data structure. We only need a small set of simple dependencies to model fairly complex deformations. We achieve this with the smart use of extra intermediary ghost nodes as temporary results, which correspond to the common subexpressions in the computations. The DAG data structure can be compared to a tree for symbolic expressions as used in many symbolic packages like *Mathematica*$^{\text{TM}}$ [21], and described in the literature of parsing and formal languages [2]. Since we allow the sharing of intermediate results through several dependencies using the same common nodes, the structure is no longer a tree. Nevertheless the DAG data structure can be seen as is a symbolic evaluator.

We now describe only three simple dependencies. They are a small subset of all the dependencies that we have implemented.

Fixed Point This dependency does not have any node as a parent. It takes 3 constants λ_1, λ_2, and λ_3 to define its position and Jacobian:

$$p_i = \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix} \quad \text{and} \quad J_i = \mathbf{0}.$$

Since the position is fixed and does not depend on any other node or parameter, the Jacobian is simply $\mathbf{0}$.

Linear Combination of Points This dependency requires a set of nodes $\{n_1, \ldots, n_k\}$ and a set of constants $\{\lambda_1, \ldots, \lambda_k\}$. Then, as the name implies:

$$p_i = \sum_{j=1}^{k} \lambda_j p_{n_j} \quad \text{and} \quad J_i = \sum_{j=1}^{k} \lambda_j J_{n_j}.$$

Add Parameterized Vector This dependency adds a vector represented by three constants λ_1, λ_2, and λ_3, scaled by parameter l, to an existing node j.

$$p_i = p_j + q_l \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix} \quad \text{and}$$

$$J_i = J_j + \begin{bmatrix} & 0 & & \lambda_1 & & 0 & \\ \cdots & 0 & \cdots & \lambda_2 & \cdots & 0 & \cdots \\ & 0 & & \lambda_3 & & 0 & \end{bmatrix}.$$

$$\underbrace{\hphantom{\cdots\ 0\ \cdots\ \lambda_2}}_{l}$$

3.4. Advantages of the DAG approach

The two main advantages of the graphical approach are flexibility and performance. On the flexibility side, the deformable model framework becomes more general, because it is easy to swap out one model for another without having to alter the framework in any other way. In addition, the framework can create multiresolution models dynamically, just by culling nodes from the DAG.

On the performance side, the DAG structure makes it easy to factor out common subexpressions and to associate them with ghost nodes. As a result, the number of computations on the deformable model is drastically reduced. It is even conceivable — although we have not implemented it — to use symbolic manipulation technologies to factor out these subexpressions automatically.

Furthermore, as PC-based architectures become more and more ubiquitous even for the most demanding computer vision applications, main memory bandwidth becomes a major performance bottleneck, because images and large models are typically too large to fit in the caches of current PC processors. As a result, the order of computations and their memory access patterns have a larger effect on performance than the raw number of floating point operations. By using topological sorting techniques, the deformable model framework can determine the optimal order and memory storage of computations automatically. In our face tracking experiments described in the next section, this optimization alone sped up the running time by more than 35% on an Intel Xeon running at 2 GHz, and by 45% on an AMD Athlon running at 1.2 GHz. This optimization would be very difficult to perform if the model were hardcoded.

4. Examples

The DAG data structure is one component of a larger system we have under development. We use this system for the two examples in this section: face tracking and cyberglove data visualization.

We use *Lua*, a fully programmable embedded language [1], to describe and construct the DAG data structure of the models. This allows a powerful and efficient way to actually create the models, and their dependencies, procedurally. Since we have a fully functional language for the model description, we can automate the model's DAG construction, and all the dependencies parameters. For example, the face model of the face (in Section 4.1) is automatically built from the position of the undeformed points and the regions of the face surface where the deformations should affect. In Section 4.2, we construct a set of affine frames to build a hand model automatically.

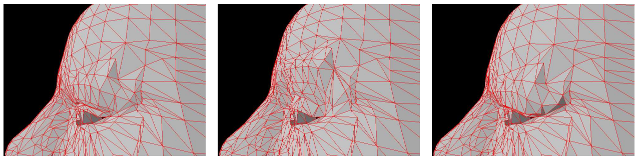

Figure 4. Three positions of the eyebrows.

4.1. Face Tracking

In this section we briefly describe how we have created a simple, yet powerful, deformable face model using the DAG data structure. We show some results of our face tracking system.

We started with a static a geometric model of a head, publicly made available by the computer graphics group of the University of Washington[3] as part of [16, 17]. We carefully extracted a mask of the face, and simplified it using `SlimKit Surface Modeling Tools`[4]. The result was a static mask model of a generic face with 1101 nodes and 2000 faces. We then marked regions in the surface that are affected by a parameter, and how this effect varied in this region. For every parameter we applied a *layer of deformation*. Each layer of deformation created one new ghost node for every surface node affected by this parameter. The ghost node took over the previous behavior of the surface node. The surface node itself was altered to encode the extra deformation in a dependency on the new ghost node. Figure 4 is an example of the effect of one single deformation (eyebrow), showing three snapshots of the movement of the eyebrows.

The model has one parameter to raise/lower eyebrows, two parameters for the smiling/stretching of the mouth, one parameter for the opening of the jaw, and another 7 for the reference frame (a total of 11 parameters). For simplicity, we modeled the the jaw movement as an affine transformation. This approximation is sufficient for small openings of the mouth, even though a rotation operation is the correct way to model the jaw. Note (in the movie) that for a large opening of the mouth the system correctly tracks the the mouth movement and does not loose track, even if the shape is temporarily not accurate.

Figure 5 has six snapshots of a tracking experiment using this model. This sequence has approximately 4000 grayscale frames captured at 60Hz on a 640×480 resolution. The tracking technique we used is partially described in [11]. It consists of a statistical technique to adaptively integrate different computer vision cues.

Before the beginning of a tracking section we perform a fit of the base mask. The fit process provides us with a new set of positions for the points (that represent the face at rest), and the orientation and translation of the model in the first frame.

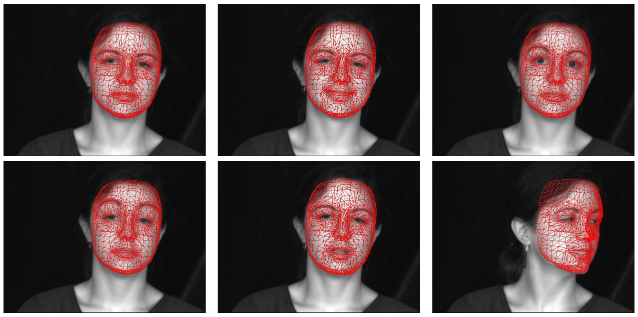

Figure 5. Snapshots of a tracking sequence of approximate 4000 frames. The deformable model used here has only 11 parameters.

4.2. Cyberglove Visualization and Calibration

Visualization is an important application in human body motion research. Deformable models are an ideal tool for this visualization, because they are powerful enough to describe the complex shape and deformations of human body parts, yet the parameterization that governs the deformations is often extremely simple. As a result, they are easy to manipulate during the visualization process. Moreover, it is possible to parameterize the models in a manner compatible with the motion capture systems that are often used for acquiring the data.

As an example, consider visualizing the human hand with data captured from a Cyberglove$^{\text{TM}}$. This instrument delivers values between 0 and 255 representing a measure of the joint angles of the finger digits, and the abduction angles between fingers, as well as wrist yaw and pitch. The raw values between 0 and 255 need to be converted to joint angles via a conversion function F_c, which is different for every human. As a result, F_C needs to be carefully calibrated, especially for those applications that are highly sensitive to slight variations in the joint angle, such as gesture and sign language recognition. To achieve this calibration, it is necessary to visualize the cyberglove data with a hand model, while adjusting the calibration coefficients carefully.

The DAG structure provides a very simple and elegant way to parameterize the hand model that we use for visualizing and calibrating the cyberglove data. The raw values from the cyberglove and the calibration coefficients both constitute the model parameters. For each joint angle, we create a set of four ghost nodes that represent the affine frame for the digit controlled by this angle. These four nodes are parameterized by the corresponding cyberglove values and calibration coefficients. We express all other nodes that belong to the

[3] `http://www.cs.washington.edu/research/graphics/projects/realface/`

[4] `http://www.cs.cmu.edu/~garland/quadrics/`

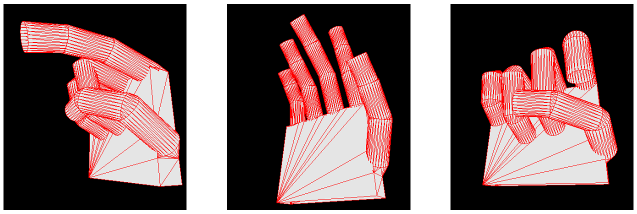

Figure 6. Three example deformations of the hand during cyberglove visualization and calibration.

same digit as barycenters in that affine frame; that is, an unparameterized weighted sum of these four nodes. Figure 6 shows three examples of visualizing and adjusting the calibration of cyberglove data.

Note that with some refinement of the hand model — through adding more nodes —, the same DAG could be used for tracking a hand with a deformable model. The tracking algorithm would then estimate the parameters of the model in a manner consistent with the values that the cyberglove generates. The compelling advantage of this consistency is that it makes it easy to switch from computer vision-based tracking to a cyberglove, and vice versa, without having to adjust the rest of the application to deal with a different set of parameters.

5. Conclusions and Future Work

In this paper we introduced a powerful data structure that can represent models for both computer vision and graphics. Using a small set of simple mathematical primitives (the DAG dependencies) it is possible to describe extremely complex deformations. In addition, new dependencies can be easily created by applying the chain rule to determine the Jacobian.

When creating hard coded models by hand, the task of computing the Jacobians is hard and extremely time consuming . Using the DAG data structure, it is easy to automatically specify complex deformations over large number of nodes – the computation of the Jacobians is done automatically, on demand.

The DAG data structure can represent both layered and hierarchical deformations as we have shown in the two examples. A possible future application is to define a deformable face whose parameters coincide with the MPEG4 facial animation parameters (FAP), thus allowing our face tracking system to generate MPEG4 data.

This technique allows the formal specification of deformations as graphs. An interesting problem, for example, that remains open is how to develop algorithms that can simplify a graph to obtain a more computationally efficient structure. Another important and extremely hard application is the construction of the dependency graph for a set of nodes based on data.

Acknowledgments

This work was supported in part by an NSF Career Award NSF-9624604, NSF EIA-98-09209, AFOSR F49620-98-1-0434, and NSBRI-NBPF00202. The first author was partially supported by CNPq - Brazilian's "*Conselho Nacional de Desenvolvimento Científico e Tecnológico*".

References

[1] Lua Website. http://www.lua.org.

[2] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley Publishing Company, 1986.

[3] A. Blake and M. Isard. *Active Contours: The Application of Techniques from Graphics,Vision,Control Theory and Statistics to Visual Tracking of Shapes in Motion*. Springer-Verlag, 1998.

[4] V. Blanz and T. Vetter. A morphable model for the synthesis of 3D faces. In *SIGGRAPH*, pages 187–194, 1999.

[5] T. Cootes and C. Taylor. Active shape models - their training and application. *Computer Vision and Image Understanding*, 61(1):38–59, 1995.

[6] J. Craig. *Introduction to Robotics: Mechanics and Control*. Addison Wesley Longman, Inc, 1989.

[7] D. DeCarlo and D. Metaxas. Optical flow constraints on deformable models with applications to face tracking. *IJCV*, 38(2):99–127, July 2000.

[8] D. DeCarlo, D. Metaxas, and M. Stone. An anthropometric face model using variational techniques. In *SIGGRAPH*, pages 67–74, 1998.

[9] L. Farkas, editor. *Antropometry of the Head and Face*. Raven Press, 1994.

[10] J. Foley, A. van Dam, S. Feiner, and J. Hughes. *Computer Graphics: Principles and Practice*. Addison Wesley Longman, Inc., 2nd edition edition, 1995.

[11] S. Goldenstein, C. Vogler, and D. Metaxas. Affine arithmetic based estimation of cue distributions in deformable model tracking. In *CVPR*, pages 1098–1105, 2001.

[12] J. Gomes, L. Darsa, B. Costa, and L. Velho. *Warping and Morphing of Graphical Objects*. Morgan Kauffman, 1999.

[13] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active Contour Models. *IJCV*, 1:321–331, 1988.

[14] D. Metaxas. *Physics-based Deformable Models: Applications to Computer Vision, Graphics and Medical Imaging*. Kluwer Academic Publishers, 1996.

[15] J. Park, D. Metaxas, and A. Young. Deformable models with parameter functions: Application to heart-wall modeling. In *CVPR*, pages 437–442, 1994.

[16] F. Pighin, J. Hecker, D. Lischinski, R. Szeliski, and D. Salesin. Synthesizing realistic facial expressions from photographs. In *SIGGRAPH*, pages 75–84, 1998.

[17] F. H. Pighin, R. Szeliski, and D. Salesin. Resynthesizing facial animation through 3d model-based tracking. In *ICCV*, pages 143–150, 1999.

[18] A. Requicha. Representation for Rigid Solids: Theory, Methods, and Systems. *ACM Computing Surveys*, 12:437–464, 1980.

[19] D. Samaras, D. Metaxas, P. Fua, and Y. Leclerc. Variable albedo surface reconstruction from stereo and shape from shading. In *CVPR*, pages 480–487, 2000.

[20] C. Sminchisescu, D. Metaxas, and S. Dickinson. Improving the Scope of Deformable Model Shape and Motion Estimation. In *CVPR*, pages 485–492, 2001.

[21] S. Wolfram. *The Mathematica Book*. Cambridge University Press, 4^{th} edition edition, 1999.