EFFICIENT FEATURE EXTRACTION FOR 2D/3D OBJECTS IN MESH REPRESENTATION

Cha Zhang and Tsuhan Chen

Dept. of Electrical and Computer Engineering, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213, USA
{czhang, tsuhan}@andrew.cmu.edu

ABSTRACT

Meshes are dominantly used to represent 3D models as they fit well with graphics rendering hardware. Features such as volume, moments, and Fourier transform coefficients need to be calculated from the mesh representation efficiently. In this paper, we propose an algorithm to calculate these features without transforming the mesh into other representations such as the volumetric representation. To calculate a feature for a mesh, we show that we can first compute it for each elementary shape such as a triangle or a tetrahedron, and then add up all the values for the mesh. The algorithm is simple and efficient, with many potential applications.

1. INTRODUCTION

3D scene/object browsing is becoming more and more popular as it engages people with much richer experience than 2D images. The Virtual Reality Modeling Language (VRML) [1], which uses mesh models to represent the 3D content, is rapidly becoming the standard file format for the delivery of 3D contents across the Internet. Traditionally, in order to fit graphics rendering hardware well, a VRML file models the surface of a virtual object or environment with a collection of 3D geometrical entities, such as vertices and polygons.

In many applications, there is a high demand to calculate some important features for a mesh model, e.g., the volume of the model, the moments of the model, or even the Fourier transform coefficients of the model. One example application is the search and retrieval of 3D models in a database [2][3][9]. Another example is shape analysis and object recognition [4]. Intuitively, we may calculate these features by first transforming the 3D mesh model into its volumetric representation and then finding these features in the voxel space. However, transforming a 3D mesh model into its volumetric representation is a time-consuming task, in addition to a large storage requirement [5][6][7].

Work supported in part by NSF Career Award 9984858.

In this paper, we propose to calculate these features from the mesh representation directly. We calculate a feature for a model by first finding it for the elementary shapes, such as triangles or tetrahedrons, and then add them up. The computational complexity is proportional to the number of elementary shapes, which is typically much smaller than the number of voxels in the equivalent volumetric representation. Both 2D and 3D meshes are considered in this paper. The result is general and has many potential applications.

The paper is organized as follows. In Section 2 we discuss the calculation of the area/volume of a mesh. Section 3 extends this idea and presents the method to compute moments and Fourier transform for a mesh. Some applications are provided in Section 4. Conclusions and discussions are given in Section 5.

2. AREA/VOLUME CALCULATION

The computation of the volume of a 3D model is not a trivial work. One can convert the model into a discrete 3D binary image. The grid points in the discrete space are called *voxels*. Each voxel is labeled with '1' or '0' to indicate whether this point is inside or outside the object. The number of voxels inside the object, or equivalently the summation of all the voxel values in the discrete space, can be an approximation for the volume of the model. However, the transforming from a 3D mesh model into a binary image is very time-consuming. Moreover, in order to improve the accuracy, the resolution of the 3D binary image needs to be very high, which can further increase the computation load.

2.1. 2D Mesh Area

We explain our approach starting from the computation of areas for 2D meshes. A 2D mesh is simply a 2D shape with polygonal contours. As shown in Figure 1, suppose we have a 2D mesh with bold lines representing its edges. Although we can discretize the 2D space into a binary image and calculate the area of the mesh by counting the pixels inside the polygon, doing so is very computationally intensive.

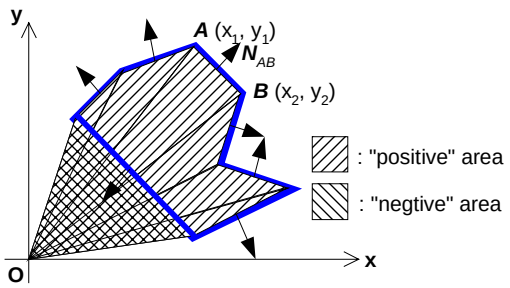

Figure 1: The calculation of a 2D polygon area

To start with our algorithm, let us make the assumption that the polygon is close. If it is not, a contour close process can be performed first [9]. Since we know all the vertices and edges of the polygon, we can calculate the normal for each edge easily. For example, edge *AB* in Figure 1 has the normal:

$$N_{AB} = \frac{-(y_2 - y_1)\hat{x} + (x_2 - x_1)\hat{y}}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}} \quad (1)$$

where (x_1, y_1) and (x_2, y_2) are the coordinates of vertices *A* and *B*, respectively, and $\hat{x}$ and $\hat{y}$ are the unit vectors for the axes. We define the *normal* here as a normalized vector which is perpendicular to the corresponding edge and pointing outwards of the mesh. In computer graphics literature, there are different ways to check whether a point is inside or outside a polygon [8], thus it is easy to find the correct direction of the normals. Later we will show that even if we only know that all the normals are pointing to the same side of the mesh (either inside or outside, as long as they are consistent), we are still able to find the correct area of the mesh.

After getting the normals, we construct a set of triangles by connecting all the polygon vertices with the origin. Each edge and the origin form an elementary triangle, which is the smallest unit for computation. We define the *signed area* for each elementary triangle as below: The magnitude of this value is the area of the triangle, while the sign of the value is determined by checking the position of the origin with respect to the edge and the direction of the normal. Take the triangle *OAB* in Figure 1 as an example. The area of *OAB* is:

$$|S_{OAB}| = \left|\frac{1}{2}(-x_2 y_1 + x_1 y_2)\right|. \quad (2)$$

The sign of S_{OAB} is the same as the sign of the inner product $\overrightarrow{OA} \cdot N_{AB}$, which is positive in this case.

The total area of the polygon can be computed by summing up all the *signed areas*. That is,

$$S_{total} = \sum_i S_i \quad (3)$$

where *i* goes through all the edges or elementary triangles.

Following the above steps, the result of equation (3) is guaranteed to be positive, no matter the origin is inside or outside the mesh. Note here that we do not make any assumption that the polygon is convex.

In real implementation, we do not need to check the signs of the areas each time. Let:

$$S'_i = \frac{1}{2}(-x_{i2} y_{i1} + x_{i1} y_{i2}),$$
$$S'_{total} = \sum_i S'_i. \quad (4)$$

where *i* stands for the index of all the edges or elementary triangles. (x_{i1}, y_{i1}), (x_{i2}, y_{i2}) are coordinates of the starting point and the end point of edge *i*. When we loop through all the edges, we need to keep forwarding so that the inside part of the mesh is always kept at the left hand side or the right hand side. According to the final sign of the result S'_{total}, we may know whether we are looping along the right direction (the right direction should give the positive result), and the final result can be simply achieved by taking the magnitude of S'_{total}.

2.2. 3D Case

We can extend the above algorithm into the 3D case. In a VRML file, the mesh is represented by a set of vertices and polygons. Before we calculate the volume, we do some preprocessing on the model and make sure that all the polygons are triangles. Such preprocessing, called triangulation, is commonly used in mesh coding, mesh signal processing, and mesh editing. The direction of the normal for a triangle can be determined by the order of the vertices and the right-hand rule, as shown in Figure 2. The consistent condition is very easy to satisfy. For two neighboring triangles, if the common edge has different directions, then the normals of the two triangles are consistent. For example, in Figure 2, *AB* is the common edge of triangle *ACB* and *ABD*. In triangle *ACB*, the direction is from *B* to *A*, and in triangle *ABD*, the direction is from *A* to *B*, thus $\boldsymbol{N}_{ACB}$ and $\boldsymbol{N}_{ABD}$ are consistent.

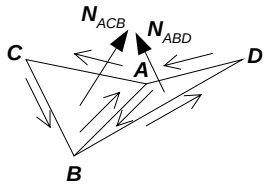

Figure 2: Normals and order of vertices

In the 3D case, the elementary calculation unit is a tetrahedron. For each triangle, we connect each of its vertices with the origin and form a tetrahedron, as shown in Figure 3.

As in the 2D case, we define the *signed volume* for each elementary tetrahedron as: The magnitude of its value is the volume of the tetrahedron, and the sign of the value is determined by checking if the origin is at the same side as the normal with respect to the triangle. In Figure 3, tri-

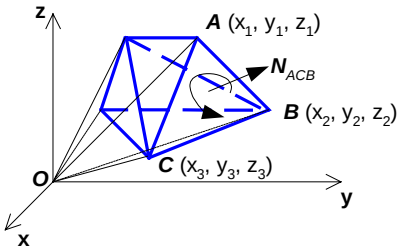

Figure 3: The calculation of 3D volume

angle *ACB* has a normal $\boldsymbol{N}_{ACB}$. The volume of tetrahedron *OACB* is:

$$|V_{OACB}| = \left|\frac{1}{6}(-x_3y_2z_1 + x_2y_3z_1 + x_3y_1z_2 - x_1y_3z_2 - x_2y_1z_3 + x_1y_2z_3)\right|. \quad (5)$$

As the origin *O* is at the opposite side of $\boldsymbol{N}_{ACB}$, the sign of this tetrahedron is positive. The sign can also be calculated by inner product $\overrightarrow{OA} \cdot N_{ACB}$.

In real implementation, again we only need to compute:

$$V'_i = \frac{1}{6}(-x_{i3}y_{i2}z_{i1} + x_{i2}y_{i3}z_{i1} + x_{i3}y_{i1}z_{i2} - x_{i1}y_{i3}z_{i2} - x_{i2}y_{i1}z_{i3} + x_{i1}y_{i2}z_{i3}). \quad (6)$$

$$V'_{total} = \sum_i V'_i$$

where *i* stands for the index of triangles or elementary tetrahedrons. (x_{i1}, y_{i1}, z_{i1}), (x_{i2}, y_{i2}, z_{i2}) and (x_{i3}, y_{i3}, z_{i3}) are coordinates of the vertices of triangle *i* and they are ordered so that the normal of triangle *i* is consistent with others. Volume of a 3D mesh model is always positive. The final result can be achieved by take the absolute value of V'_{total}. In order to compute other 3D model features such as moments or Fourier transform coefficients, we reverse the sequence of vertices for each triangle if V'_{total} turns out to be negative.

3. MOMENTS AND FOURIER TRANSFORM

The above algorithm can be generalized to calculate other features for 2D and 3D mesh models. Actually, *whenever the feature to be calculated can be written as a signed sum of features of the elementary shape (triangle in the 2D case and tetrahedron in the 3D case), and the feature of the elementary shape can be derived in an explicit form, the proposed algorithm applies.* Although this seems to be a strong constrain, many of the commonly-used features fall into this category. For example, all the features that have the form of integration over the space inside the object can be calculated with this algorithm. This includes moments, Fourier transform, wavelet transform, and many others.

In classical mechanics and statistical theory, the concept of moments is used extensively. In this paper, the moments of a 3D mesh model are defined as:

$$M_{pqr} = \iiint x^p y^q z^r \rho(x,y,z)\,dxdydz \quad (7)$$

where $\rho(x,y,z)$ is an indicator function:

$$\rho(x,y,z) = \begin{cases} 1, & \text{if } (x,y,z) \text{ is inside the mesh} \\ 0, & \text{otherwise.} \end{cases} \quad (8)$$

and *p*, *q*, *r* are the orders of the moment. Central moments can be obtained easily from the result of equation (7). Since the integration can be rewritten as the sum of integrations over each elementary shape:

$$M_{pqr} = \sum_i s_i \iiint x^p y^q z^r \rho_i(x,y,z)\,dxdydz \quad (9)$$

where $\rho_i(x,y,z)$ is the indicator function for elementary shape *i*, and s_i is the sign of the *signed volume* for shape *i*.

We can use the same process as that in Section 2 to calculate a number of low order moments for triangles and tetrahedrons that are extensively used. A few examples for the moments of a tetrahedron are given in the Appendix. More examples can be found in [9].

Fourier transform is a very powerful tool in many signal processing applications. The Fourier transform of a 2D or 3D mesh model is defined by the Fourier transform of its indicator function:

$$\Theta(u,v,w) = \iiint e^{-i(xu+yv+zw)} \rho(x,y,z)\,dxdydz \quad (10)$$

Since Fourier transform is also an integration over the space inside the object, it can also be calculated by decomposing the integration into integrations over each elementary shape. The explicit form of the Fourier transform of a tetrahedron is given in the Appendix.

As the moments and Fourier transform coefficients of an elementary shape are explicit, the above computation is very efficient. The computational complexity is $\boldsymbol{O(N)}$, where *N* is the number of edges or triangles in the mesh. Note that in the volumetric approach, where a 2D or 3D binary image is obtained first before getting any of the features, the computational complexity is $\boldsymbol{O(M)}$, where *M* is the number of grid points inside the model, not considering the cost of transforming the data representation. It is obvious that *M* is typically much larger that *N*, especially when a relatively accurate result is required and the resolution of the binary image has to be large. The storage space required by our algorithm is also much smaller.

Previous work by Lien and Kajiya [10] provide a similar method for calculating the moments for tetrahedrons. Our work gives more explicit forms of the moments and extends their work to calculating the Fourier transform.

4. APPLICATIONS

A good application of our algorithm is to find the principal axes of a 3D mesh model. This is useful when we want to compare two 3D models that are not well aligned. In a 3D

model retrieval system [2][9], this is required because some of the features may not be invariant to arbitrary rotations.

We construct a 3x3 matrix by the second order moments of the 3D model:

$$S = \begin{bmatrix} M_{200} & M_{110} & M_{101} \\ M_{110} & M_{020} & M_{011} \\ M_{101} & M_{011} & M_{002} \end{bmatrix}. \quad (11)$$

The principal axes are obtained by computing the eigenvectors of matrix *S*, which is also known as the principle component analysis (PCA). The eigenvector corresponding to the largest eigenvalue is made the first principal axis. The next eigenvector corresponding to the secondary eigenvalue is the second principal axis, and so on. In order to make the final result unique, we further make sure that the 3rd order moments, M_{300} and M_{030}, are positive after the transform. Figure 4 shows the results of this algorithm.

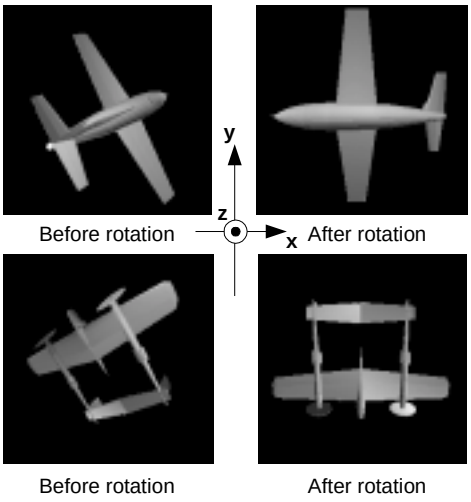

Figure 4: 3D models before and after PCA

The Fourier transform of a 3D mesh model can be used in many applications. For example, the coefficients can be directly used as features in a retrieval system [9]. Other applications are shape analysis, object recognition, and model matching. Note that in our algorithm, the resulting Fourier transform is in continuous form. There is no discretization alias since we can evaluate a Fourier transform coefficient from the continuous form directly.

5. CONCLUSIONS AND DISCUSSIONS

In this paper, we propose an algorithm for computing features for a 2D or 3D mesh model. Explicit methods to compute the volume, moments and Fourier transform from a mesh representation directly are given. The algorithm is very efficient, and has many potential applications.

The proposed algorithm still has some room for improvement. For example, it is still difficult to get the explicit form of a high order moment for a triangles and tetrahedrons. Also the Fourier transform may lose its computational efficiency if many coefficients are required simultaneously. More research is in progress to speed this up.

REFERENCES

[1] R. Carey, G. Bell, and C. Marrin, "The Virtual Reality Modeling Language". Apr. 1997, *ISO/IEC DIS 14772-1*. [Online]: http://www.web3d.org/Specifications/.

[2] Eric Paquet and Marc Rioux, "A Content-based Search Engine for VRML Database", *Computer Vision and Pattern Recognition, 1998. Proceedings. 1998, IEEE Computer Society Conference on*, pp. 541-546, 1998.

[3] Sylvie Jeannin, Leszek Cieplinski, Jens Rainer Ohm, Munchurl Kim, *MPEG-7 Visual part of eXperimentation Model Version 7.0, ISO/IEC JTC1/SC29/WG11/N3521*, Beijing, July 2000.

[4] Anthony P. Reeves, R. J. Prokop, Susan E. Andrews and Frank P. Kuhl, "Three-Dimensional Shape Analysis Using Moments and Fourier Descriptors", *IEEE Trans. Pattern Analysis and Machine Intelligence*, pp. 937-943, Vol. 10, No. 6, Nov. 1988.

[5] Homer H. Chen, Thomas S. Huang, "A Survey of Construction and Manipulation of Octrees", *Computer Vision, Graphics, and Image Processing*, pp. 409-431, Vol. 43, 1988.

[6] Shi-Nine Yang and Tsong-Wuu Lin, "A New Linear Octree Construction by Filling Algorithms", *Computers and Communications, 1991. Conference Proceedings. Tenth Annual International Phoenix Conference on*, pp. 740-746, 1991.

[7] Yoshifumi Kitamura and Fumio Kishino, "A Parallel Algorithm for Octree Generation from Polyhedral Shape Representation", *Pattern Recognition, 1996. Proceedings of the 13th International Conference on*, pp. 303-309, Vol. 3, 1996.

[8] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes, *Computer Graphics principles and practice, Second Edition*, Addison-Wesley Publishing Company, Inc., 1996.

[9] http://amp.ece.cmu.edu/projects/3DModelRetrieval/.

[10] Sheue-ling Lien and James T. Kajiya, "A Symbolic Method for Calculating the Integral Properties of Arbitrary Nonconvex Polyhedra", *IEEE Computer Graphics and Applications*, pp. 35-41, Oct. 1984.

APPENDIX

$$M_{000} = \frac{1}{6}\left(-x_3y_2z_1 + x_2y_3z_1 + x_3y_1z_2 - x_1y_3z_2 - x_2y_1z_3 + x_1y_2z_3\right).$$

$$M_{100} = \frac{1}{4}\left(x_1 + x_2 + x_3\right)M_{000}.$$

$$M_{200} = \frac{1}{10}\left(x_1^2 + x_2^2 + x_3^2 + x_1x_2 + x_2x_3 + x_1x_3\right)M_{000}.$$

$$M_{300} = \frac{1}{20}\left(x_1^3 + x_2^3 + x_3^3 + x_1^2(x_2 + x_3) + x_2^2(x_1 + x_3) + x_3^2(x_1 + x_2) + x_1x_2x_3\right)M_{000}$$

$$\Im(u,v,w) = M_{000}^{*}$$

$$\left(\frac{i * e^{i(ux_1+vy_1+wz_1)}}{(ux_1+vy_1+wz_1)(ux_1-ux_2+vy_1-vy_2+wz_1-wz_2)(ux_1-ux_3+vy_1-vy_3+wz_1-wz_3)}\right.$$

$$+\frac{i * e^{i(ux_2+vy_2+wz_2)}}{(ux_2+vy_2+wz_2)(-ux_1+ux_2-vy_1+vy_2-wz_1+wz_2)(ux_2-ux_3+vy_2-vy_3+wz_2-wz_3)}$$

$$\left.+\frac{i * e^{i(ux_3+vy_3+wz_3)}}{(ux_3+vy_3+wz_3)(-ux_1+ux_3-vy_1+vy_3-wz_1+wz_3)(-ux_2+ux_3-vy_2+vy_3-wz_2+wz_3)}\right)$$

$$-\frac{i}{(ux_1+vy_1+wz_1)(ux_2+vy_2+wz_2)(ux_3+vy_3+wz_3)}$$