Sumit Basu, Kentaro Toyama, and Alex Pentland. "A Consistent Method for Function Approximation in Mesh-based Applications." MIT Media Laboratory Vision and Modeling Technical Report #486. January, 1999.

A Consistent Method for Function Approximation in Mesh-based Applications

Sumit Basu*, Kentaro Toyama†, and Alex Pentland*
*MIT Media Laboratory and †Microsoft Research
{sbasu,sandy}@media.mit.edu, kentoy@microsoft.com
http://www.media.mit.edu/~sbasu

Abstract

Most mesh-based tracking/modeling techniques face a point at which they need to approximate the underlying image function with a smoother, more regular function. This often occurs when gradients need to be computed. The typical approach is to convolve the image with a fixed kernel. This has a number of disadvantages. First, a fixed kernel does not account for the different levels of detail in different parts of the mesh - small facets can be oversmoothed while large facets are undersmoothed. Also, the size of the tracked image may be changing dramatically in scale (as in head tracking), and thus a fixed-size kernel again presents problems. We present a method to approximate a function with a set of basis functions that are consistent with the mesh (planar triangular patches above each facet). The resulting approximation is piecewise analytic (with C0 continuity) and inherently at the same level of detail as the mesh. We show how this approximation can be very efficiently computed, touching each pixel only once, and how multiscale representations can be computed with minimal cost. We describe techniques and applications for image coding, gradient computation, tracking, and adaptive remeshing. Examples of image functions, object meshes, and the resulting approximations are shown.

1 Introduction

Computer vision is replete with a variety of mesh-based techniques - deformable templates (see [3]), physics-based vision (e.g., [1]), analysis by synthesis (e.g., [5]), model-based coding, etc. Most of these methods depend on meshes built up of triangular facets. These meshes are often constructed to accurately model the level of detail - areas with complex geometry/texture have many small facets, while smooth areas have fewer and larger facets. This allows the physics of the mesh to accurately represent the physics of the body, because the level of physics complexity is *consistent* with the geometry. However, these mesh-based methods reach a point when this consistency is lost. That comes when the methods need to derive information from an approximation of the image function underlying the mesh, typically in the form of gradients. Because the image itself is noisy and local derivatives are noisier still, the general practice is to smooth the image with a fixed-size kernel. Because the facets and the underlying level of detail are of varying sizes, though, a fixed kernel oversmooths in some places and undersmooths in others. Furthermore, even though a particular size of kernel can work reasonably well when the mesh is at a particular scale (i.e., its projection covers a particular number of pixels in the image), when the object being tracked changes in scale (i.e. moves forward or back), the projection of the mesh becomes smaller or larger and the fixed kernel is no longer the right size.

In this paper, we present a method for approximating an image function in a manner that is consistent with the mesh. Our method uses a function basis that comes from the finite element domain and is consistent with the geometry and complexity of a given triangular mesh. This has a number of strong features. First, it is detailed where the mesh is detailed, and coarse where the mesh is coarse. Next, the resulting approximated function is C0 continuous (no step functions) and in a piecewise analytic form. The approximation can be found very efficiently, touching each pixel only once. Furthermore, we will show how multiscale versions of the function can be found with minimal computation. We will also show how the mesh can be adaptive to the function, adding or removing nodes where necessary.

We first describe the origins of the function basis and its form. We then show how to approximate a function with this basis, going from the continuous domain to an account of how to compute things efficiently with real (i.e., discrete) images. We then show how multiscale representations can be formed with this method. We end with some applications, examples, and future directions.

2 The Function Basis

As we described earlier, the typical meshes used in computer vision are made up of flat, triangular elements. In finite elements, this corresponds to a thin

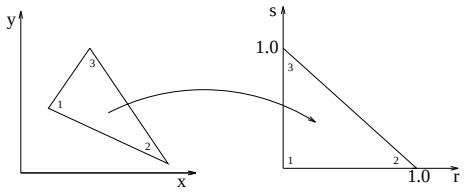

Figure 1. The mapping from world coordinates to parametric coordinates

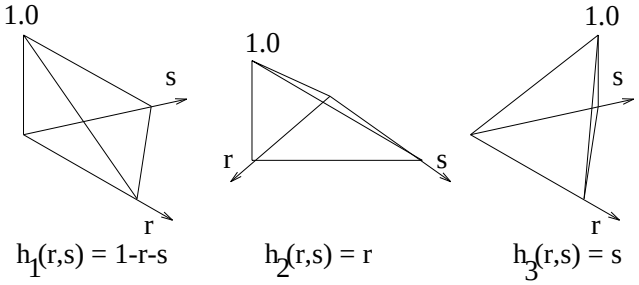

Figure 2. The interpolation functions of the isoparametric triangular element

shell structure made up of *isoparametric* elements (see the appendix of [1] for more details on the physics formulation of these meshes, and [2] for background on the finite element method and isoparametric elements). The "parametric" part of the term comes from the fact that the elements are described in a parametric coordinate system (see figure 1). The "iso" part comes from the fact that both the geometry and the underlying functions (stress, strain, fluid flow, an image, etc.) are interpolated by the same set of *interpolation functions* (also known as shape functions). These functions can be represented as a matrix, $H(r,s)$, that takes the function values at the nodes and produces the function value at the parametric location (r,s). For the flat triangular element, the shape functions are r, s, and $1-r-s$ (see figure 2). This means we can find a function value $f(r,s)$ given the values F_1,F_2, and F_3 at the nodes (see numbering in figure 1) in the following way:

$$f(r,s) = \begin{bmatrix} 1-r-s & r & s \end{bmatrix} \begin{bmatrix} F_1 \\ F_2 \\ F_3 \end{bmatrix} \qquad (1)$$

where the first term is $H(r,s)$. The geometry can be represented in the same way, replacing F_i with X_i, Y_i, and Z_i the locations of the vertices. This then maps every point in (r,s) space to the actual location in (x,y,z). Note that this simple interpolant preserves the function values at the nodes and interpolates linearly between them, resulting in simple, flat triangular facets for the geometry (as desired). This representation for a function is thus a "natural" one for the meshes used in vision applications. Since the function representation is completely determined by the values of the nodes, there is one basis function per node. Furthermore, since the value at a given node influences the function in all the facets touching that node, the basis functions look like the one shown in figure 3.

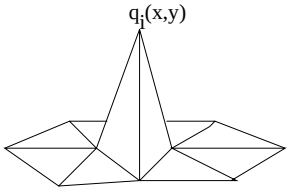

Figure 3. The basis function at node i

Our task is thus to find the the values of F_i for all the nodes i in the mesh. However, we cannot simply sample the values of the function at the nodes - this would ignore most of the underlying image and give suboptimal results. We wish to find the set of function values that minimize the error between our approximation and the underlying function. The next section describes how we find these values.

3 Computing the Approximation

As we described in the previous section, we now have a basis of i functions, one for each node, all living in the *domain* of the mesh, which is the area of the image that is covered by the mesh. Note that we only have a 2D mesh to deal with at this point - once the mesh has been projected onto the camera view, we do not need to deal with the depth dimension. We are thus trying to approximate the image on this 2D domain using the basis functions. We will call the basis function involving node i as $q_i(x,y)$. The problem we wish to solve is thus finding α_i such that we minimize the error E:

$$E = \int\int (f(x,y) - \sum_i \alpha_i q_i(x,y))^2 dxdy \qquad (2)$$

where the integral is taken over the domain of the mesh. Let us now consider the domain as being an infinite-dimensional vector space (see [8, 7] for more about this perspective). We can then rewrite our problem in the following representation:

$$Q\alpha = f \qquad (3)$$

where the columns of Q are the basis functions q_i. Since the vector space is infinite-dimensional and we have a finite basis (with i entries), this is an overconstrained problem. Since we want to minimize the squared error (equation 2), the solution is identical to the familiar least-squares case:

$$\alpha = \prec Q, Q \succ^{-1} \prec Q, f \succ \qquad (4)$$

Where $\prec a, b \succ$ denotes the inner product between $a(x,y)$ and $b(x,y)$, which is defined in this vector space as the following integral over the domain:

$$\prec a, b \succ = \int\int a(x,y)b(x,y)dxdy \tag{5}$$

The result of such an inner product is simply a scalar, so when we apply equation 4, we end up with a numerical matrix equation for α.

Fortunately, the basis functions all have compact support (they are non-zero over small regions, see figure 3), so the integrals are all over small regions as well. Furthermore, we do not need to compute nearly as many integrals as equation 4 implies. We can compute $\prec Q, Q \succ$ without doing a single integral.

To see how this is possible, we need to go back to the interpolation equations for the geometry and rewrite the mapping from (r,s) to (x,y) as follows:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} X_2 - X_1 & X_3 - X_1 \\ Y_2 - Y_1 & Y_3 - Y_1 \end{bmatrix} \begin{bmatrix} r \\ s \end{bmatrix} + \begin{bmatrix} X_1 \\ Y_1 \end{bmatrix} \tag{6}$$

where the matrix multiplying $[rs]'$ is J, the Jacobian relating (r,s) space to (x,y) space (note that the determinant of this Jacobian, $|detJ|$, is the scaling factor for the area from (r,s) space to (x,y) space, and is thus twice the area of the triangle). An integral of a function f over the actual triangle in (x,y) space can then be equivalently written in (r,s) space as follows:

$$\int\int f(x,y)dxdy = \int\int f(r,s)drds|detJ| \tag{7}$$

Note that the basis functions' triangular patches over each facet can be simply represented as r, s, and $1-r-s$ as in equation 1 above (see also figure 2). Furthermore, since $|detJ|$ is constant, we can take it outside the integral. Now consider the integrals we need to compute - they are the inner products of each basis function q_i with each other basis function q_j. Let us split this into two cases: where $i \neq j$ and where $i = j$. In this first case, the only places where the inner product $q_i(r,s)q_j(r,s)$ is nonzero is where the two corresponding nodes share one or more facets. For each such facet in the domain, the integral over the facet is:

$$\int\int rsdrds|detJ| = \frac{|detJ|}{24} \tag{8}$$

Note that we could have chosen the product $r(1-r-s)$ or $s(1-r-s)$ and would find the same result. This is necessary because our choice of node numberings was arbitrary - any product of two shape functions for two different nodes will have this integral over the facet. We thus compute the entry $Q_{i,j} = Q_{j,i}$ by summing up $|detJ|/24$ for all the overlapping facets in the basis functions for node i and node j.

In the case where $i = j$, we have a similar situation, but now all the facets overlap. Furthermore, the integral for each facet is now:

$$\int\int r^2drds|detJ| = \frac{|detJ|}{12} \tag{9}$$

We thus compute the entry $Q_{i,i}$ by summing up $|detJ|/12$ for each facet in the basis function for node i. We can make this computation even more efficient by precomputing both the triangle areas and the mapping between nodes and the facets which are attached to them, preventing the need to iterate over all of the facets for each Q entry.

We do need to compute integrals for the second term, $\prec Q, f \succ$, but we can do this quite efficiently since the basis functions have compact support. Specifically, this means that for each q_i, the inner product function $q_i(r,s)f(r,s)$ is non-zero only over the facets touching node i. As a result, each integral necessary for $\prec Q, f \succ$ is quite small - it is the sum of the integrals over these facets. However, note that since the basis functions overlap, we will be iterating over each facet multiple times. We can avoid this by considering $\prec Q, f \succ$ in terms of the facets. The integral of the inner product of the function with interpolation function p (where p = 1,2, or 3) for a given facet contributes to the $\prec Q, f \succ$ entry for the node corresponding to node p for this facet. We can thus compute the integral with all three interpolation functions at the same time, visiting each pixel only once. We then iterate over the facets (only once), accumulating the entries of $\prec Q, f \succ$ in this manner.

The next question is how to compute these integrals in terms of the pixels. The typical approach in finite elements is to take a fixed sampling in (r,s) space [2] due to the difficulty of inverting the shape functions in the general case. This approach has the failing of getting the same number of samples from every triangle, regardless of size. From the perspectives of both efficiency and information content, we would like to iterate uniformly over the pixels in (x,y) space. Fortunately, in the case of the flat triangular element, we can easily invert the shape functions from equation 6:

$$\begin{bmatrix} r \\ s \end{bmatrix} = \frac{1}{|detJ|} \begin{bmatrix} Y_3 - Y_1 & X_1 - X_3 \\ Y_1 - Y_2 & X_2 - X_1 \end{bmatrix} \begin{bmatrix} x - X_1 \\ y - Y_1 \end{bmatrix} + \begin{bmatrix} X_1 \\ Y_1 \end{bmatrix} \tag{10}$$

This maps the values in (x,y) for a particular facet to (r,s). Thus, to numerically compute an inner prod-

uct integral of an interpolation function $h(r,s)$ with the target function, we first rewrite the interpolation function in (x,y) space as

$$h(x,y) = h(r(x,y), s(x,y)) \tag{11}$$

and then iterate over the pixels, approximating the integral with a summation:

$$\int\int h(r(x,y), s(x,y)) f(x,y) dx dy \tag{12}$$

$$\approx \sum_{facet} h(r(x,y), s(x,y)) f(x,y) \tag{13}$$

$$\tag{14}$$

We have now seen how to compute both $\prec Q, Q \succ$ and $\prec Q, f \succ$, which allows us to solve for α, the approximated function values at every node. We then have a piecewise analytic representation of the function over the facets.

4 Efficiency and Accuracy Considerations

We have already seen how we can compute $\prec Q, Q \succ$ analytically and $\prec Q, f \succ$ with a minimum of computation (in fact, fewer than with a convolution of any reasonably-sized kernel). One expensive step that we have not yet discussed is the computation of $\prec Q, Q \succ^{-1}$. This is an n by n matrix, where n is the number of nodes in the mesh (often quite large). Therefore, this inversion is quite expensive. Unfortunately, every time the mesh deforms, $\prec Q, Q \succ$ changes and thus its inverse changes as well. To be exact, we would have to re-invert the matrix with every mesh deformation. Remember, though, that this term encodes the overlap of the facets in the mesh. Since the topology of the mesh is not changing, the overlap will not change very much with reasonable deformations of the mesh. Our preliminary experiments have shown that we can approximate functions accurately without recomputing $\prec Q, Q \succ^{-1}$. There are some cases that would change $\prec Q, Q \succ$ drastically. The first of these is uniform scaling of the mesh. This is easy to deal with, though, since it ends up scaling $\prec Q, Q \succ$ by a constant and thus its inverse by the reciprocal. The second case is when there is a change in the 3D pose of the model. This will significantly change areas and change the basis set as well, since new facets will come into view while others are culled out of view. One approach to these situations is to precompute $\prec Q, Q \succ$ and its inverse for a range of views. Another approach is to never find $\prec Q, Q \succ^{-1}$ explicitly but instead to solve $\prec Q, Q \succ \alpha = \prec Q, f \succ$ iteratively with a method such as the biconjugate gradient algorithm [6], which works well here since $\prec Q, Q \succ$ is nearly diagonal (it has a very limited bandwidth due to the compact support of the basis functions). Furthermore, since we know the topology of the mesh, we can be very efficient about the order we choose to solve the resulting equations (i.e., how we traverse the nodes in the computation). This method is preferable where the geometry/basis is changing dramatically from frame to frame.

Another consideration is the accuracy to which equation 14 approximates the integrals over the facets. If a facet only covers one and a half pixels, the computed integrals can have significant errors (though the overall contribution to $\prec Q, f \succ$ will be minimal because of its small size with respect to the domain). We can increase the accuracy of the integrals by oversampling the image, a common practice in computer graphics (as in antialiasing).

A last issue concerns the use of the resulting function approximation. We have a piecewise analytic form over the facets, but if we want to evaluate the function at a point (x,y), we first need to find which facet it is in. This can be an expensive process. A simple way around this is keeping a reverse mapping of (x,y) to facet numbers. This is simply an image of the same size as the function, where each entry is the number of the facet of that covers that point (or -1 if no such facet exists). We can "render" this image while going through the pixels in our computation of $\prec Q, Q \succ$.

5 Techniques and Examples

In this section, we describe several useful techniques with the resulting function approximation. This includes image coding, finding function gradients, computing multiscale representations of a function, and adapting a function representation (by adapting the underlying mesh). We have included a few preliminary figures, and are in the process of applying these techniques to several other problems.

5.1 Basic Examples and Image Coding

Figure 4 shows an example of a target function (an intensity image), a mesh, and the approximation of the function on that mesh. The mesh has 206 nodes and has a domain of 6580 pixels for the image shown. The mean absolute error per pixel using this approximation was 1.021 units (the original grayscale image had a range from 0 to 255). As expected, the approximation smooths the image, but now in the desired way - as opposed to a fixed kernel, it preserves detail where the mesh is detailed, and smooths it where the mesh is coarse. Furthermore, since the resulting description of the function is extremely compact (one value per node), this technique could be very useful for image

coding. This may have great potential given that most model-based coding techniques depend on mesh representations.

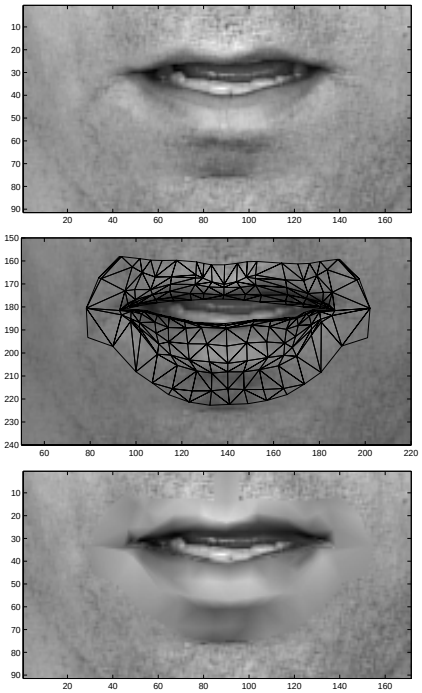

Figure 4. An example of approximating an image function with a mesh: the original image, the image with the mesh overlaid, and the image with the mesh domain approximated by our method.

The next set of figures (figures 6 and 7 show the effects of approximating a function with a series of increasingly detailed meshes. Figure 6 shows the original image, while figure 7 shows the meshes used (level 0, level 1, and level 2) and the resulting approximations. Note that the eye and lip regions were not approximated by the mesh, since for accurate reconstruction, a model developed specifically for these areas should be used (as in figure 4). The number of nodes and the mean absolute error for each mesh are shown in table 1 below.

This approximation was for a domain of 55,948 pixels. This further demonstrates the potential of our technique for image coding - at the level 0 mesh, this is a compression factor of 708.2; at the level 1 mesh, a factor of 255.5; and even at the level 2 mesh, it is a factor of 74.5. Furthermore, if the mesh were subdivided adaptively (we propose a method for doing this in a later section), it should be possible to achieve the same error with even fewer nodes.

Table 1. Number of nodes and mean absolute error (per pixel) for face image reconstruction example

Mesh Level	Number of Nodes	MAE (per pixel)
0	79	3.80
1	219	2.29
2	751	1.45

5.2 Computing Gradients

As we stated in the introduction, one of the motivations for this function approximation technique was to be able to take gradients that would be "smoothed" in a manner consistent with the mesh. Because our approximation is linear within each facet, taking the gradient within a facet is trivial:

$$\begin{bmatrix} \frac{df}{dx} \\ \frac{df}{dy} \end{bmatrix} = \begin{bmatrix} \frac{F_2 - F_1}{X_2 - X_1} \\ \frac{F_2 - F_1}{Y_2 - Y_1} \end{bmatrix} \quad (15)$$

We thus have the gradients for the function over the entire domain of the mesh at minimal cost.

5.3 Multiscale Representation

Another common need in computer vision and image coding applications is representing functions at multiple levels of detail. Typically this is done in the image space with Gaussian pyramids [4] and related techniques (wavelets, etc.). The problem with this approach for mesh-based methods is that we typically want to change the spatial extent of the approximation *in terms of the mesh* and not in terms of the image. In other words, we don't want to double the spatial extent everywhere (as going to a new level in the pyramid implies), but to go from small facets to large facets and from large facets to even larger facets. A mesh topology with which this is particularly convenient is a *subdivision mesh*, in which each facet at a higher level of detail is made by splitting the lower level of detail facet into two or more facets. An instance of this method splitting each facet into three more is shown in figure 5.

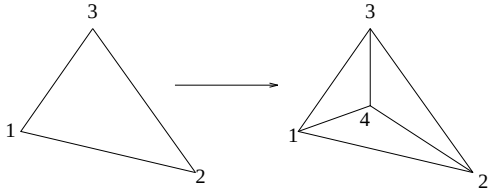

Figure 5. The subdivision of a facet

We then start out with the mesh at the finest of detail we are interested in (i.e., the highest level of subdivision). To then go to the next coarser level (combining

sets of three facets into 1), we need to find the inner product integrals with a new set of basis functions for the new $\prec Q, f \succ$. However, since the smaller facets share boundaries with the new, larger facets (see figure 5), the larger basis functions are linear over the smaller facets as well. Furthermore, since any linear function $b_{lin}(r,s)$ over a facet can be represented exactly as combination of the three interpolation functions, i.e.,

$$b_{lin}(r,s) = B_1(1-r-s) + B_2 r + B_3 s \qquad (16)$$

the inner product integral with the function can be written as

$$\int\int f(r,s) b_{lin}(r,s) dr ds \qquad (17)$$

$$\begin{aligned} = & \int\int f(r,s)(B_1(1-r-s) \\ & + B_2 r + B_3 s) dr ds \\ = & B_1 \int\int f(r,s)(1-r-s) dr ds \\ & + B_2 \int\int f(r,s) r dr ds \\ & + B_3 \int\int f(r,s) s dr ds \end{aligned} \qquad (18)$$

Where the integrals over (r,s) in equation 18 have already been computed (and presumably stored) when finding $\prec Q, f \succ$ for the original, smaller facets. As a result, we can compute the function approximation at the coarser level from the detailed level *without touching the function values again*. Furthermore, we only need to do three multiplies per small facet.

5.4 Mesh Adaptation

When going from the coarse level of the function approximation to a higher level of detail (i.e., subdividing further than our original mesh), there is no such simplification if we want an exact solution. We need to redo all of the integrals with all of the facets. However, if we are willing to accept an approximate solution (that we can refine), we can not only do this efficiently, but also adaptively, subdividing some facets of the mesh but not others based on the complexity of the underlying function. This brings us to our last application - adding complexity to a mesh based on the complexity of the function.

The basis for deciding whether to subdivide a facet is a matter of choice, but the simplest one is to look at the squared error between the target function and the approximation over the facet. A threshold can then be defined - either absolute, in terms of the error value, or relative, in terms of the overall squared error - above which the facet will be subdivided. Once the subdivision is made, a new node (and thus a new basis function) is introduced into the domain. Note that the support for this new node's basis function does not overlap with any facets other than the one being subdivided. However, the nodes of the original facet now have different support, which is why an exact solution requires significant computation.

An approximate solution, though, is easy to find. Let us assume for the moment that the node values for the rest of the mesh retain their optimality even when we add the new node. Then all we have to do is find the function approximation value for this node j. We then want to minimize the following error, where all the α_i's are held constant and the only degree of freedom is α_j:

$$\int\int (f(x,y) - (\sum_{i \neq j} \alpha_i q_i(x,y) + \alpha_j q_j(x,y)))^2 dx dy$$

$$\int\int ((f(x,y) - \sum_{i \neq j} \alpha_i q_i(x,y)) - \alpha_j q_j(x,y)))^2 dx dy$$

The second form puts us back into the familiar framework from equation 2, where are now trying to approximate the function $f - \sum \alpha_i q_i$ with one basis function, q_j. Going through the same development as earlier, this leads us to the following least squares solution:

$$\alpha_j = \frac{\prec q_j, f(x,y) - \sum_{i \neq j} q_i(x,y) \succ}{\prec q_j, q_j \succ} \qquad (19)$$

Since our assumption that the other α's are all at their optimal values even after the new node is introduced is false, the new approximation will not be least-squares optimal in terms of our new basis. However, we are clearly guaranteed the squared error will be less than before the subdivision. Furthermore, we can refine our solution by applying the method in equation 19 to the nodes surrounding the new node, and then the nodes surrounding these nodes, and so on. This will propagate the effects of adding the new node and move the α's towards their optimal values.

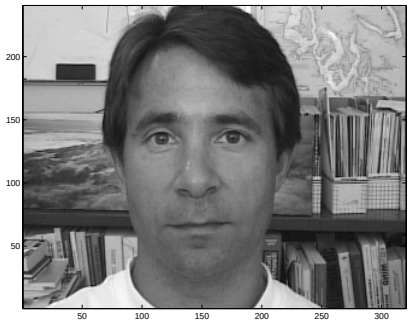

Figure 6. Original Image

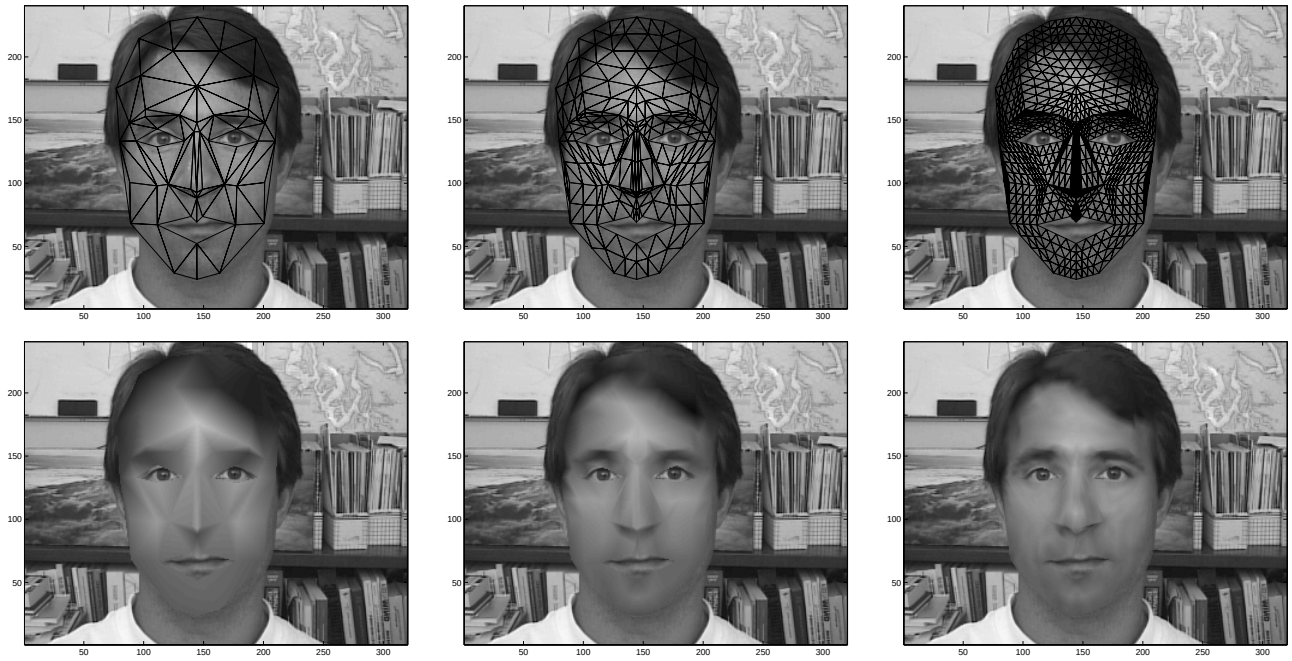

Figure 7. Level 0, 1, and 2 mesh and resulting approximation. Note that the eye and lip regions were not approximated by the mesh.

6 Conclusions and Future Directions

We have presented a method for approximating a function underlying a mesh that is consistent with the mesh. We have shown how this approximation has a number of advantages over smoothing with a fixed kernel and other image-domain approximation techniques: its basis is the natural one implied by flat triangular elements, it preserves level of detail in the function to the same degree as the geometry of the mesh, and it can be efficiently computed (touching each pixel only once). Furthermore, exact multiscale representations can be found with minimal computation (three multiplies per facet), and adaptive (though inexact) remeshing is possible with few computations as well.

We are currently working on employing this approximation method for a variety of computer vision tasks, including image coding (as shown in figure 7), tracking intensity images (computing gradients in time difference images with the approximated function), maximizing model probabilities (computing gradients in likelihood images), adaptively remeshing range data, and fitting one mesh to another (e.g., fitting a low polygon-count model to range data of a head). We also hope to further explore the use of iterative solution techniques to avoid ever having to compute $\prec Q, Q \succ^{-1}$.

7 Acknowledgements

Thanks to Jacob Strom for suggesting the application of this technique to image coding.

References

[1] S. Basu, N. Oliver, and A. Pentland. "3D Lip Shapes from Video: A Combined Physical-Statistical Model". *Speech Communication*, 26:131–148, 1998.

[2] K.-J. Bathe. *Finite Element Procedures*. Prentice-Hall, 1996.

[3] E. Blake and A. Yuille. *Active Vision*. MIT Press, 1992.

[4] P. Burt and E. H. Adelson. The laplacian pyramid as a compact image code. *IEEE Transactions on Communication*, COM-31:532–540, 1983.

[5] H. Li, P. Roivainen, and R. Forchheimer. 3-d motion estimation in model-based facial image coding. *PAMI*, 15(6):545–555, June 1993.

[6] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipies in C*. Cambridge University Press, 1992.

[7] G. Strang. *An Introduction to Applied Mathematics*. Cambridge Press, 1986.

[8] G. Strang and T. Nguyen. *Wavelets and Filter Banks*. Wellesley-Cambridge Press, 1996.