Highly Parallel Algorithms for Visual-Perception-Guided Surface Remeshing

Lianping Xing, Xiaoting Zhang, Charlie C.L. Wang, and Kin-Chuen Hui ■ *Chinese University of Hong Kong*

A proposed framework extracts visual-perception information in a polygonal model's image space and maps it back to the Euclidean space. On the basis of these cues, the framework generates a saliency field to resample the input model. A projection operator further optimizes the distribution of resampled points.

Computer graphics applications often employ polygon meshes to represent 3D geometric shapes. Methods to create meshes include using modeling software and 3D range scans. However, owing to these methods' limitations, the resulting meshes' quality might be unsatisfactory, even if they capture 3D shapes accurately. Some meshes might even contain defects such as gaps, holes, and self-intersecting triangles.

Remeshing is usually employed to improve the geometry's quality and the mesh's connectivity. Researchers have devised many remeshing approaches that generate a point distribution capturing the underlying model's characteristics. These approaches produce different patterns (for example, through uniform sampling, curvature-adapted sampling, and Poisson disk sampling). However, most of them use sequential algorithms and take tremendous time to compute. They also have difficulty exploiting GPUs' computational power.

To improve remeshing, we propose exploiting human visual-perception cues. Research has shown that you can greatly enhance complex 3D models' comprehensibility by guiding users' attention to visually salient regions.[1] Owing to visual-perception techniques' efficient visual persuasion in traditional art and technical illustration, they've been widely used in many computer graphics applications, including feature extraction and shape matching.[2] However, until now, no visual-perception-guided remeshing approach has been available.

To remedy that situation, we developed an effective, efficient remeshing framework (see Figure 1). It generates quality meshes in three steps: visual-feature extraction, resampling, and sample optimization and meshing. All the algorithms in our framework can be easily parallelized to run on GPUs.

Good Remeshing

Depending on the target application, a remeshing approach's goal might vary. However, any remeshing approach should have four properties.

First, it should be general. It shouldn't place strict requirements on the input models' quality. You should be able to apply it to a variety of models, such as orientable two-manifold piecewise linear surfaces and polygon soups.

Second, it should be accurate. It should generate a mesh that's as close as possible to the input model. Also, the vertices' distribution on the mesh should lead to a good element shape (for example, nearly regular triangles). To achieve high accuracy, the vertices usually must be distributed adaptively according to some density function.

Third, it should be efficient. It should be able to process huge models with a massive number of polygons in a reasonable amount of time.

Finally, it should be simple—that is, easy to implement.

Published by the IEEE Computer Society 0272-1716/14/$31.00 © 2014 IEEE

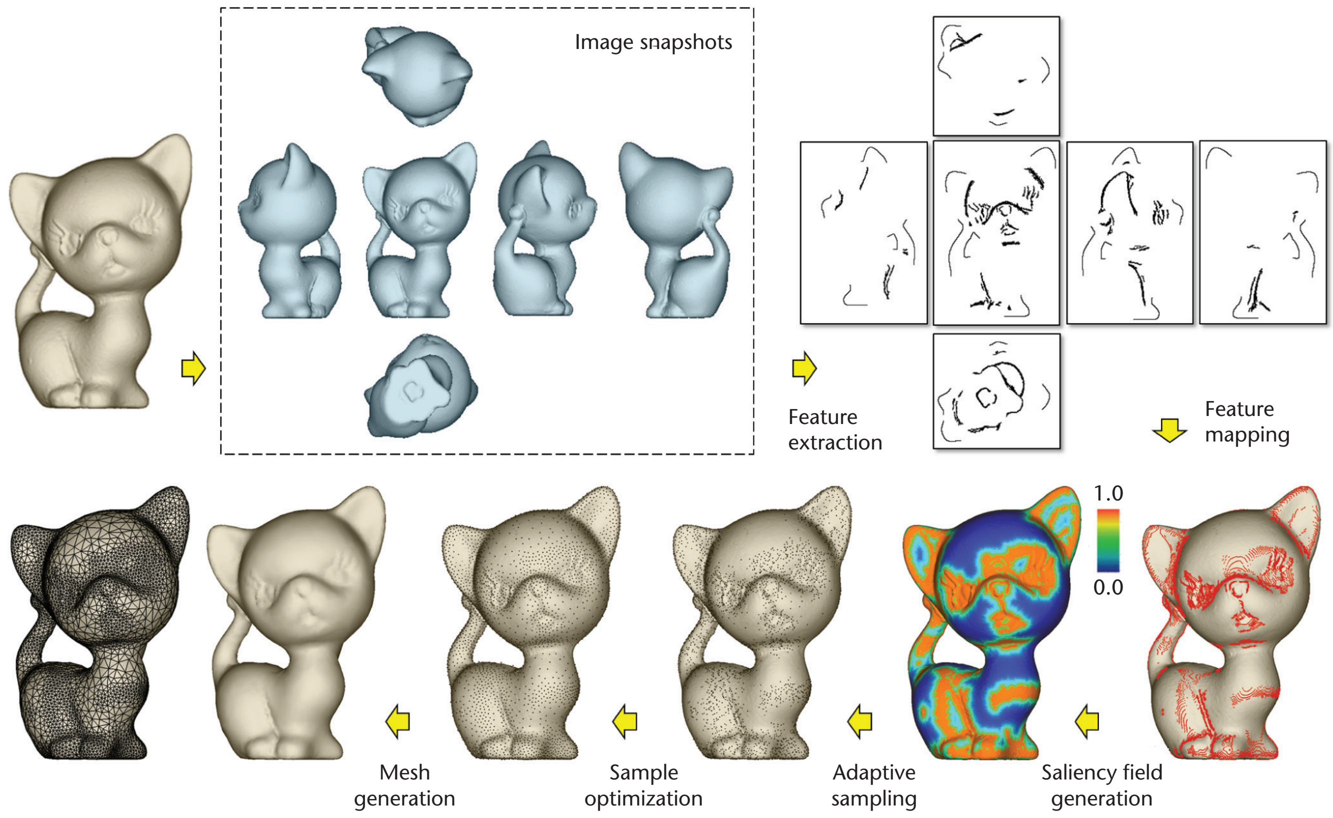

Figure 1. Our visual-perception-guided remeshing framework. First, we capture an input model's image snapshots in six orthogonal views. We then extract the perceptual features in the image space and map them back onto the input model as saliency points (see the red ones on the model at the end of the bottom row). After that, we generate a saliency field and use it to govern adaptive sampling. Finally, we use adaptive weighted locally optimal projection (AWLOP) operators to optimally position the sample points, which we connect to create a two-manifold mesh surface.

Extracting Perception Cues

A number of excellent approaches depict 3D shapes in different styles according to visual requirements. Advanced line-drawing techniques can effectively depict 3D shapes and match the effectiveness of artists' drawings.[3] It's commonly agreed that a good depiction of 3D shape should include a wealth of visual cues beyond contours. In this sense, a given model's perception cues aren't limited by its silhouettes. Some approaches follow this research thread.[2] However, they don't process polygon soup models.

Here, we borrow tools from computer vision to extract the perception cues in the image space (that is, the different views of input models). We map the results back to 3D models as 3D visual-saliency points.

Visual-Saliency Extraction

We start remeshing by taking snapshots of an input model. To obtain a mesh that produces good visualization results, the snapshots capture six orthogonal views (see Figure 1). The images can be efficiently obtained through a hardware-accelerated graphics pipeline (for example, OpenGL).

Preprocessing for problematic models. Our framework processes models with holes and other topological problems. If we didn't process the holes' boundaries, the holes would be treated simply as small features. To fill them, we apply a low-pass filter—specifically, a median filter using a $k \times k$ aperture. We choose k according to the input model's noise level; for a highly noisy model, we use a larger k. In our tests, $k = 7$ worked well. Figures 2a and 2b show this preprocessing's results. Before applying the median filter, we must convert an RGB image into a grayscale image.

After preprocessing, we extract each snapshot's visual saliency, using an inner-feature filter and a silhouette-feature filter.

The inner-feature filter. This filter spans a Gaussian over the image. The grayscale value of each pixel (u, v) is $\widehat{P}(u,v)$. The filter adopts an $n \times n$ 2D Gaussian mask

$$G(i,j) = \alpha e^{\frac{\left(i-\frac{n-1}{2}\right)^2+\left(j-\frac{n-1}{2}\right)^2}{(2\sigma)^2}},$$

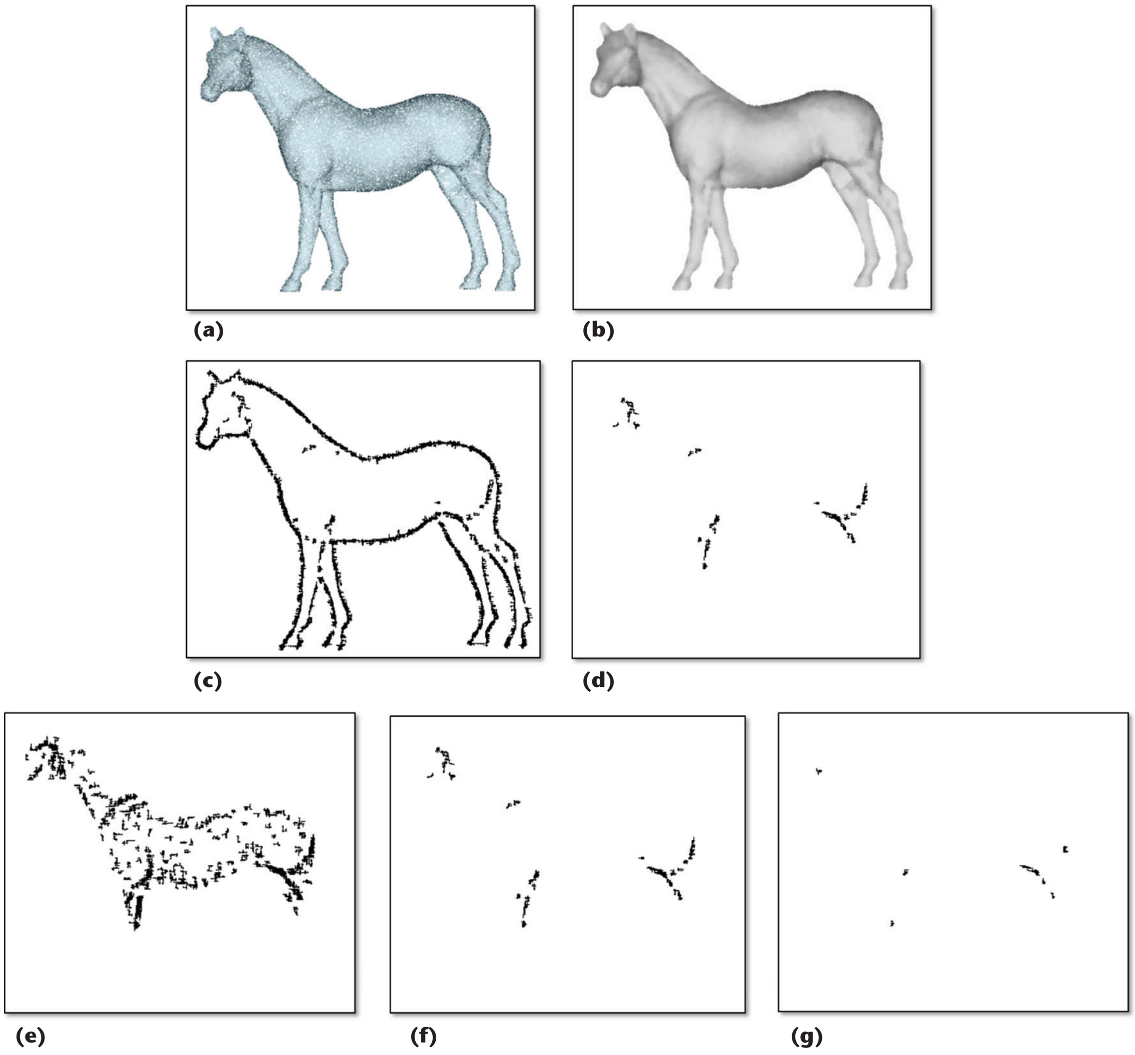

Figure 2. Preprocessing and the inner-feature filter. During preprocessing, we (a) took a problematic horse model and (b) applied a median filter to fill its holes. Applying the inner-feature filter to the grayscale image in Figure 2b produced (c) a binary image containing (d) inner features. By applying different values for a threshold *C*, we extracted different numbers of features: (e) *C* = 0, (*f*) *C* = 3, and (*g*) *C* = 6.

where the bandwidth parameter $\sigma = 0.3\left(\left((n-1)/2\right)-1\right)+0.8$, α is a scale factor chosen so that $\Sigma_{i,j\in mask}G(i,j) = 1$, and *e* means exponent. Our tests used $n = 11$ and $\alpha = 0.0242$.

Using this mask, we obtain a weighted average image, $T(u, v)$:

$$T(u,v) = \sum_{i,j\in mask} G(i,j)\hat{P}(u+i,v+j) - C,$$

where *C* is a threshold controlling how many feature edges can be extracted. We obtain the resulting binary image containing the candidate features' pixels:

$$\bar{F} = \left\{(u,v) \mid T(u,v) > \hat{P}(u,v)\right\}.$$

If we use a smaller value for *C*, more pixels will remain after filtering. Our implementation used *C* = 3. We determine the final inner features by excluding the silhouette pixels.

Figures 2c and 2d illustrate applying the inner-feature filter to a horse model; Figures 2e through 2g compare the results for different values of *C*.

The silhouette-feature filter. The silhouette's thin-and-sharp features (for example, the horse model's ears and legs) are important for representing a 3D model's shape. Using the silhouette-feature filter, we extract these pixels and add them to $\bar{F}$. These features are foreground pixels that

- have background pixels in their 8-neighbors and
- are a relatively short distance from the input model's skeleton.

So, to find these features, we extract the skeleton of an input model in the image space and compute

–1	–1	–1	–1	–1
0	0	0	0	0
2	2	2	2	2
0	0	0	0	0
–1	–1	–1	–1	–1

(a)

–1	0	2	0	–1
–1	0	2	0	–1
–1	0	2	0	–1
–1	0	2	0	–1
–1	0	2	0	–1

(b)

0	–1	–1	0	2
–1	–1	0	2	0
–1	0	2	0	–1
0	2	0	–1	–1
2	0	–1	–1	0

(c)

2	0	–1	–1	0
0	2	0	–1	–1
–1	0	2	0	–1
–1	–1	0	2	0
0	–1	–1	0	2

(d)

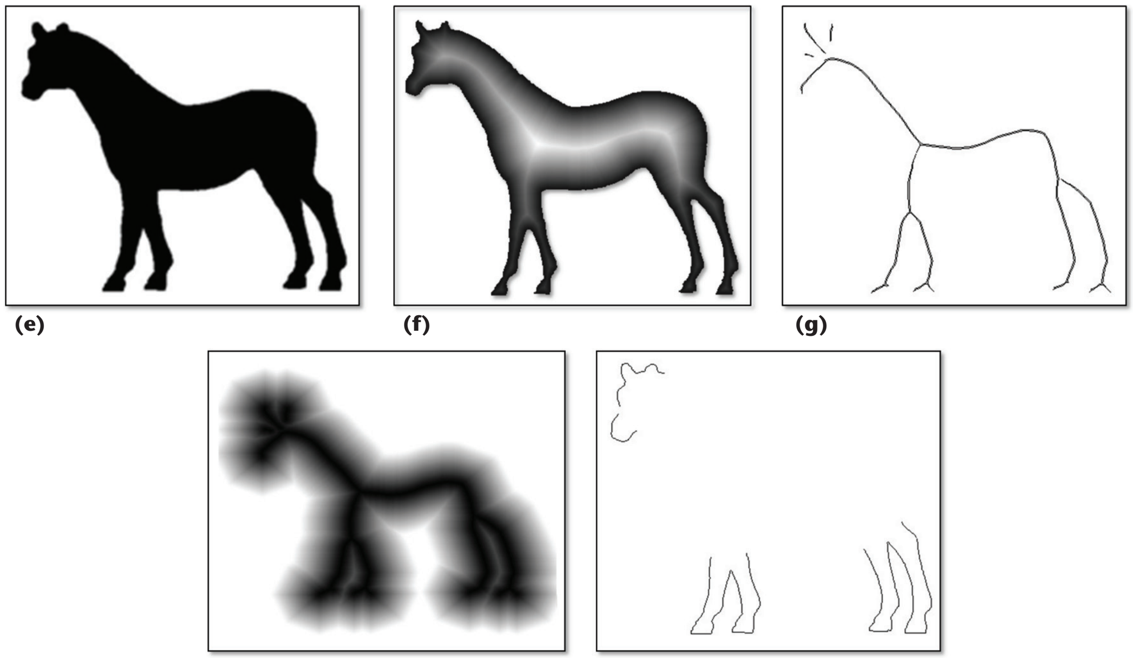

Figure 3. The silhouette-feature filter. To extract a skeleton on a silhouette distance map, we use (a) horizontal, (b) vertical, (c) 45-degree, and (d) 135-degree edge extraction kernels. For (e) an input model in the image space, we use (f) the distance map of its silhouette to extract (g) its skeleton. From the skeleton, we obtain (h) a distance map with which we detect (i) the model's thin-and-sharp features on the silhouette.

distances from the silhouette pixels to the skeleton. These two tasks execute in parallel with the help of a highly parallel distance transformation algorithm.

The distance transformation uses a small mask, M, to propagate the distances over the image iteratively. At the transformation's beginning, we assign zero to the distances D at the source pixels, whereas we initialize the distances at the other pixels as infinity. Then, the distance value at each pixel (u, v) updates in parallel:

$$D(u,v) = \min_{(i,j)\in M} \left\{ D^{prev}(u+i, v+j) + D_M(i,j) \right\},$$

where D^{prev} is the distance at a pixel in the previous iteration, and $D_M(i, j)$ is the local distance from (i, j) to a mask's center. If we use a $z \times z$ mask,

$$D_M(i,j) = \left(\left(i - \frac{z-1}{2} \right)^2 + \left(j - \frac{z-1}{2} \right)^2 \right)^{1/2}.$$

The update runs in parallel on all pixels for a few iterations until no distance value changes, which we can easily check using scan primitives. Or, we can conduct a fixed number, m, of iterations. We set m as the input image's diagonal length divided by the size z of M. We could use a more accurate parallel distance transformation algorithm. However, because accuracy isn't a major concern in our filter, we employ the previous algorithm, which is easier to implement.

To extract the skeleton, we first compute a distance map from every foreground pixel to the silhouette pixels (by using the silhouette pixels as sources in the distance transformation algorithm). We apply four kernels (see Figures 3a through 3d) across the distance map to extract the corresponding directional gradients. If we find a significantly large gradient at a pixel, we consider the pixel as belonging to the skeleton.

Specifically, if $K_i(u, v)$ denotes the response of kernel i for (u, v), we use the filter to extract the skeleton pixels $\bar{S}$:

```
Input: the set of saliency points $\widehat{F}$ and a model $M$
Output: the propagated depths on vertices of $M$
Set the depth values of all points in $\widehat{F}$ as zero;
Set the depth values of all vertices on $M$ as $\infty$;
Insert all the points of $\widehat{F}$ into the front $L$;
while $L \neq \emptyset$ do
  foreach $v_i \in M$ in parallel do
    if ($v_i$ is untraveled) AND ($v_i$ is the neighbor of a traveled point) then
      Set $v_i$ as a candidate vertex of 'next-front';
    end
  end
  Compact all 'next-front' vertices on $M$ into a new set $L'$;
  foreach $v_k \in L'$ in parallel do
    $d_{v_k} \Leftarrow \min_j\{d_{v_j}+1\}$ for all neighbors $v_j$ of $v_k$;
    Set $v_k$ as travelled;
  end
  Update the front as $L \Leftarrow L'$;
end
return;
```

Figure 4. The algorithm for parallel propagation of saliency fields. The depth d_{v_i} of every vertex v_i indicates the shortest distance to that vertex's nearest saliency point.

$$\bar{S} = \left\{(u,v) \mid \max_{\forall i}\left\{K_i(u,v)\right\} > \lambda\right\},$$

with λ = 8 being the threshold for selecting the significantly large gradient. Using a larger λ generates sparse points for the skeleton; a smaller λ gives a dense skeleton with many unwanted branches. Figures 3e through 3g illustrate applying this skeleton-extraction filter.

We generate another distance map by using the skeleton pixels as sources, the value of which presents the feature size. With this information, we can detect the silhouette pixels with a small feature size (fewer than 20 in our tests on 512 × 512 images). We add these pixels to the set of feature points. Figures 3g through 3i show the result of extracting such thin-and-sharp features on the horse model.

A highly parallel implementation on a GPU. Implementing these image-processing operators is easy. Before applying them, we evaluate them independently on the basis of the neighboring pixels' information. So, they're realized as kernels running on the GPU with the help of the CUDA (Compute Unified Device Architecture) software development kit (SDK) library.

Image Space to Euclidean Space Mapping

After extracting the visual saliency, we map the set of feature pixels back to the Euclidean space to guide sampling. To obtain efficient mapping, we employ a hardware-accelerated graphics pipeline (OpenGL in our implementation). When taking the snapshots, we record every pixel's z-buffer value. These values help to unproject every pixel in the set of feature points back to $\Re^3$ to serve as saliency points.

Visual-Perception-Guided Sampling

After generating the saliency points, we build a scalar saliency field over the entire surface domain to govern resampling.

Saliency Field Generation

We aim to distribute a user-specified number of samples over the mesh surface such that more points will be on the visually salient regions. Specifically, vertices near the saliency points should have higher visual importance. A value of 1.0 represents the highest visual importance. We assign values within [0, 1) to all vertices.

To generate the visual-saliency field, we use an advancing-front method. This method progressively moves a front L from the saliency points to their nearest-neighbor vertices on the surface and then to farther vertices until it has travelled all the vertices. Before the propagation, we insert the input model's saliency points and vertices into a k-d tree to conduct the approximate nearest-neighbor search.

To govern the propagation, we construct the neighborhood table and copy it to the GPU side. During the advancing, we update the depth d_{v_i} of every vertex v_i, which indicates the shortest distance to that vertex's nearest saliency point. We initialize the vertices' depths as $+\infty$ and set the saliency points' depths to zero. Figure 4 shows the pseudocode for parallel propagation.

Then, we set the visual importance of every vertex v_i as

$$I_{v_i} = \beta^{d_{vi}},$$

where $\beta \in [0,1]$. The larger β is, the smoother the field is. For the models in this article, we used $\beta = 0.7$ and set the number of neighbors as $k = 10$ to balance speed and accuracy.

Adaptive Sampling

We integrate the saliency field over the surface and obtain a visual quantity V_s:

$$V_s = \sum_{i=1}^{n} V_i ,$$

where n is the number of triangles. V_i is the visual quantity of the ith triangle:

$$V_i = \frac{1}{3} A_i \sum_{j=1}^{3} I_{v_j} ,$$

where A_i is the triangle area and I_{v_j} is the vertices' visual importance.

Suppose we plan to generate n_s sample points on the input model. We calculate the number of samples, n_i, in the ith triangle as $n_i = \lfloor (V_i/V_s) n_s + 0.5 \rfloor$. We round n_i to an integer, which introduces a signed quantization error E_r. As the sampling proceeds triangle by triangle, this error accumulates and can't be neglected. So, we correct the number of samples in the ith triangle to

$$n_i = \left\lfloor \frac{V_i}{V_s} n_s + \sum_{r=1}^{i-1} E_r + 0.5 \right\rfloor \qquad (1)$$

by considering the quantization error

$$E_r = \frac{V_r}{V_s} n_s - n_r$$

on all previously sampled triangles.

Uniform triangle sampling. We generate the samples in a triangle T with vertices v_i, v_j, and v_k with the help of barycentric coordinate $\mathbf{b} = (b_i, b_j, b_k)$, with $b_i, b_j, b_k \in [0, 1]$ and $b_i + b_j + b_k \equiv 1$.

First, we generate two random numbers: $r_1 \in [0, 1]$ and $r_2 \in [0, 1]$. We use them to form the barycentric coordinate:

$$\mathbf{b} = \left(\min\{r_1, r_2\}, \max\{r_1, r_2\} - \min\{r_1, r_2\}, 1 - \max\{r_1, r_2\} \right),$$

so that $\mathbf{b}$ represents a new sample point's position in the triangle. When the number of expected samples in a triangle is small (for example, $n_i < 5$), we employ this simple uniform triangle sampling because the distribution of samples according to the visual importance is contributed mainly by Equation 1.

However, when n_i becomes big, the distribution should follow the visual importance evaluated on the vertices. Then, we employ adaptive triangle sampling.

Adaptive triangle sampling. Because we know the visual-importance values I_{v_i}, I_{v_j}, and I_{v_k} on v_i, v_j, and v_k, we formulate the expected distribution of samples as a normalized function $J(\mathbf{b})$, using $\mathbf{b} = (b_i, b_j, b_k)$ and $I(\mathbf{b}) = b_i I_{v_i} + b_j I_{v_j} + b_k I_{v_k}$:

$$J(\mathbf{b}) = \frac{1}{\int_{\mathbf{T}} I(\mathbf{b})\, d\mathbf{T}} I(\mathbf{b}) ,$$

where $\int_{\mathbf{T}} I(\mathbf{b})\, d\mathbf{T} = (A/3)(I_{v_i} + I_{v_j} + I_{v_k})$, and d means derivative. For a sample point following $J(\mathbf{b})$, we

After generating the saliency points, we build a scalar saliency field over the entire surface domain to govern resampling.

obtain b_i and b_j one by one. We later determine $b_k = 1 - b_i - b_j$.

By introducing a marginal density of b_i,

$$J_M(b_i) = \int_{\mathbf{T}} J(b_i, b_j, 1 - b_i - b_j)\, db_j ,$$

we formulate the cumulative density function (CDF) of b_i as

$$F_1(x) = \int_0^x J_M(b_i)\, db_i .$$

According to Wolfgang Hörmann and his colleagues' analysis, for a CDF $F(...)$ of a random variable x, if another random variable r comes from a uniform distribution in [0, 1], the random variable $z = F^{-1}(r)$ comes from the same distribution of x.[4] Specifically, for a random draw $r_1 \in [0, 1]$, x follows the expected distribution of b_i when $F_1(x) = r_1$ is enforced. That is, we determine $b_i = x$ by solving

$$\hat{a}x^3 + \hat{b}x^2 + \hat{c}x = (I_{v_i} + I_{v_j} + I_{v_k}) r_1 ,$$

with $\hat{a} = I_{v_j} + I_{v_k} - 2I_{v_i}$, $\hat{b} = 3(I_{v_i} - I_{v_j} - I_{v_k})$, and $\hat{c} = 3(I_{v_j} + I_{v_k})$. Given the value of b_i, the conditional distribution of b_j is

$$J_C(b_j) = J(b_i, b_j, 1 - b_i - b_j)/J_M(b_i).$$

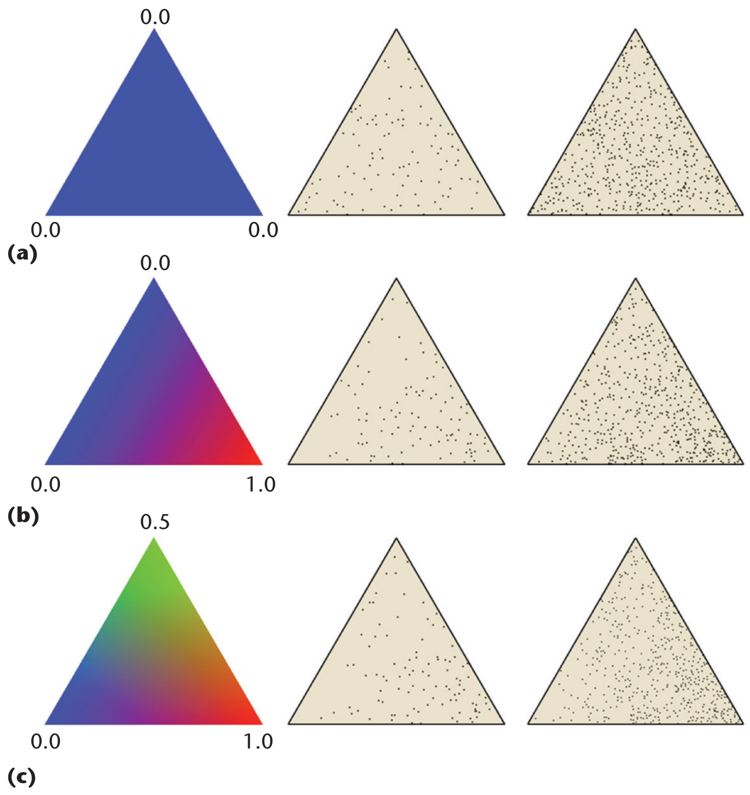

Figure 5. Generating adaptive samples. (a) We employ an inversion method[3] according to the visual importance assigned to a triangle's vertices. (b) 100 samples are generated on triangles. (c) 500 samples are generated.

Then, we formulate the CDF of b_j:

$$F_2(y) = \int_0^y J_C(b_j)\,db_j \,.$$

Similarly, for a random draw $r_2 \in [0, 1]$, y follows the expected distribution of b_j when $F_2(y) = r_2$ is enforced—that is, the solution of

$$\lambda x^2 + \gamma x = \widehat{f}\,,$$

with $\lambda = (1/2)\left(I_{v_j} - I_{v_k}\right)$, $\gamma = I_{v_i}b_i - I_{v_k}b_i + I_{v_k}$, and

$$\widehat{f} = \frac{1}{2}\Big(\left(I_{v_j} + I_{v_k} - 2I_{v_i}\right)b_i^2 + 2\left(I_{v_i} - I_{v_j} - I_{v_k}\right)b_i + I_{v_j} + I_{v_k}\Big)r_2\,.$$

The sampling we described before efficiently generates samples such that the region with high visual importance has more points, which follows the extracted perception cues. As Figure 5 shows, we can generate an adaptive distribution of samples by following the visual importance assigned at vertices.

A hybrid CPU/GPU implementation. To implement the sampling in a hybrid CPU/GPU manner, we first evaluate how many points to sample on each triangle on the CPU. Then, we classify the triangles into those to sample uniformly and those to sample adaptively. Finally, we generate the samples for each group in parallel on the GPU and insert them in a 1D array with the help of the GPU's atomic operator.

A Sample Distribution Optimization

The generated samples follow the indication presented by the saliency field. However, the samples aren't distributed regularly enough to be linked to form well-shaped triangles (with nearly equal angles on vertices). To improve the distribution's regularity, we employ an iterative method. After repositioning the sample points, we use the Tight Co-Cone algorithm (available at www.cse.ohio-state.edu/%7Etamaldey/cocone.html) to connect them to form a two-manifold triangular mesh.

AWLOP

Our algorithm to reposition sample points is akin to the projection operators used in point-sample surfaces—specifically, the locally optimal projection (LOP) operator. Given a data point set $P = \{\mathbf{p}_j\} \subset \Re^3$, LOP projects a set of particles $X = \{\mathbf{x}_i\} \subset \Re^3$ onto the surface formed by P by approximating their L^1 medians.

To improve the projected particles' regularity, Hui Huang and her colleagues presented a weighted locally optimal projection (WLOP) operator, which introduces a repulsion term to control the particle distribution.[5] This pushes the particles so that their distances from their neighbors are nearly equal. In other words, this obtains a uniform distribution.

However, WLOP doesn't consider the saliency field. So, we extended it to an adaptive version—AWLOP—that incorporates the projected points' visual-saliency values. Specifically, we obtain P from M by

- uniformly sampling M if its number of vertices is small or
- directly using the vertices of M if their number is large.

The particles in X are the sample points we obtained earlier. We move every particle $\mathbf{x}_i$ to a new position. Similarly to WLOP, the position update consists of two terms. The first attracts the particle to the given point set by the weighted local density,

$$o_j = 1 + \sum_{\mathbf{p}_l \in (P \setminus \{\mathbf{p}_j\})} \theta\left(\left\|\mathbf{p}_j - \mathbf{p}_l\right\|\right).$$

The second term repulses the particles away from other particles by a particle distribution density, w_q. The updated position of $\mathbf{x}_i$ is

$$\mathbf{x}_i = \sum\nolimits_{\mathbf{p}_j \in P} \mathbf{p}_j \frac{\theta\left(\left\|\mathbf{x}_i - \mathbf{p}_j\right\|\right)/o_j \left\|\mathbf{x}_i - \mathbf{p}_j\right\|}{\sum\nolimits_{\mathbf{p}_j \in P} \theta\left(\left\|\mathbf{x}_i - \mathbf{p}_j\right\|\right)/o_j \left\|\mathbf{x}_i - \mathbf{p}_j\right\|}$$
$$+\mu \sum\nolimits_{\mathbf{x}_q \in (X \setminus \{\mathbf{x}_i\})} \left(\mathbf{x}_i - \mathbf{x}_q\right) \frac{w_q\, \theta\left(\left\|\mathbf{x}_i - \mathbf{x}_q\right\|\right)/\left\|\mathbf{x}_i - \mathbf{x}_q\right\|}{W},$$

where

$$W = \sum\nolimits_{\mathbf{x}_q \in (X \setminus \{\mathbf{x}_i\})} w_q\, \theta\left(\left\|\mathbf{x}_i - \mathbf{x}_q\right\|\right)/\left\|\mathbf{x}_i - \mathbf{x}_q\right\|, \|...\|$$

is the L^2 norm, and $\theta(r) = e^{-16r^2/h^2}$.[5] $\theta(r)$ is a rapidly decreasing smooth weight function, with the support radius h defining the influenced neighborhood's size. $\mu \in [0, 0.5)$ and h are two user-selected parameters for tuning the operator's performance.

Unlike WLOP, AWLOP defines w_q at $\mathbf{x}$ as a function of the visual importance $I(\mathbf{c}_q)$ at the closest vertex of $\mathbf{x}_q$, where $\mathbf{c}_q \in M$. To allow visual-saliency-guided particle distribution, the regions with high visual importance should have weaker repulsion forces. So, we make w_q inversely proportional to $I(\mathbf{x}_q)$:

$w_q = 1/I(\mathbf{c}_q)$.

In our tests, we set $\mu = 0.45$ and $h = 2L_{\text{avg}}$, with L_{avg} being the data points' average distance to their k-nearest neighbors. We chose $k = 20$ to balance speed and robustness.

Implementation Details and Comparisons

We implemented AWLOP in a hybrid CPU/GPU program. We constructed and updated the neighborhood table of points in P and X on the CPU, using the ANN (Approximate Nearest Neighbor) library (available at www.cs.umd.edu/~mount/ANN). The iterative updating of the particles' positions executed on the GPU. This let us efficiently optimize the samples to a regular distribution that could adapt according to the visual-saliency field.

Figure 6 compares results for LOP, WLOP, and AWLOP. For the input hand model (see Figure 6a) and its saliency field (see Figure 6b), we generated 10k points with an irregular distribution (see Figure 6c). The LOP and WLOP distributions (see Figures 6d and 6e) tended to be uniform no matter what the initial set was. AWLOP generated particles that adapted to the visual saliency (see Figures 6f and 6g).

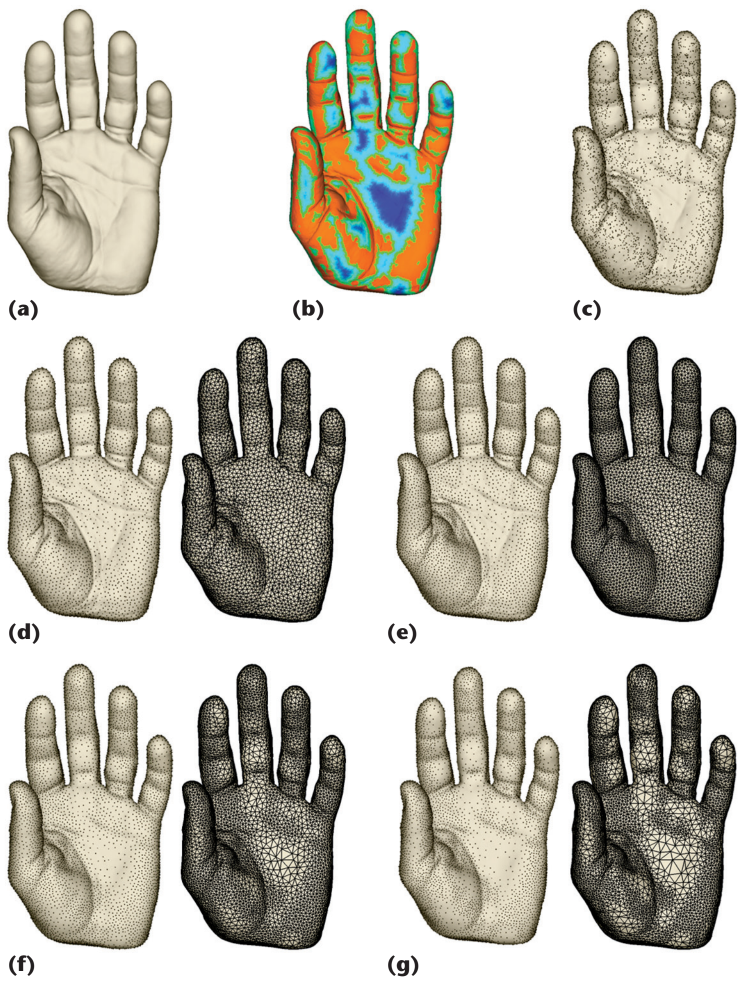

Figure 6. Comparing locally optimal projection (LOP), weighted LOP (WLOP), and adaptive WLOP (AWLOP). For (a) a hand model and (b) its visual-saliency field, our sampling method generated (c) 10k points with an irregular distribution. The results for (d) LOP and (e) WLOP exhibited uniformly distributed particles. With AWLOP, we obtained a regular and adaptive distribution with (f) $w_q = 1/I(\mathbf{c}_q)$ and (g) $w_q = 1/I^2(\mathbf{c}_q)$, where w_q is the particle distribution density and $I(\mathbf{c}_q)$ is the visual importance at the closest vertex of particle $\mathbf{x}_q$.

We set w_q as $1/I(\mathbf{c}_q)$ for Figure 6f and $1/I^2(\mathbf{c}_q)$ for Figure 6g. The distribution in Figure 6g was more adaptive than the one in Figure 6f. However, it could have led to too sparse regions when the number of particles was very small. So, our all later tests used $1/I(\mathbf{c}_q)$.

Results and Discussion

We implemented our framework using C++ and the CUDA SDK library (you can access the implementation at www2.mae.cuhk.edu.hk/%7Ecwang/GPURemeshByVisualCues.html). We ran it on a PC with an Intel Core i7 3.4-GHz CPU, 8 Gbytes of RAM, and a GeForce GTX 660 Ti GPU. All the tasks ran in parallel. Because our algorithms are highly parallel, the remeshing of all the examples took less than 10 seconds.

Related Work in Remeshing

Remeshing improves mesh quality in many computer graphics applications—for example, shape editing, animation, and numerical simulation. Recently, it has received considerable attention, and researchers have developed a variety of remeshing algorithms. Current approaches either compute the remeshing in parametric domains or directly generate it on 3D surfaces.

Parameterization-based approaches partition a parameter domain into sets of adjacent elements with the same specific properties. Xianfeng Gu and his colleagues' approach cuts the surface into patches, parameterizes it using a signal-adapted technique, and represents the surface as a set of images that store the geometry and other attributes used for visualization.[1] Vitaly Surazhsky and his colleagues' remeshing algorithm is based on local parameterization.[2] However, parameterizing free-form models is challenging and introduces severe distortions.

To alleviate these problems, researchers have proposed remeshing that directly samples 3D meshes. Greg Turk presented retiling that applies attraction-repulsion particle relaxation to resample an input mesh.[3] Yongwei Miao and his colleagues introduced curvature-aware adaptive sampling,[4] which can help produce high-quality meshes. Simon Fuhrmann and his colleagues devised curvature-adaptive remeshing based on weighted centroidal Voronoi tessellation.[5] However, none of these approaches takes visual perception into consideration. In addition, they are highly time-consuming and can't be sped up by GPU-based computing.

References

1. X. Gu, S.J. Gortler, and H. Hoppe, "Geometry Images," *ACM Trans. Graphics*, vol. 21, no. 3, 2002, pp. 355–361.
2. V. Surazhsky, P. Alliez, and C. Gotsman, "Isotropic Remeshing of Surfaces: A Local Parameterization Approach," *Proc. 12th Int'l Meshing Roundtable*, 2003, pp. 215–224.
3. G. Turk, "Re-tiling Polygonal Surfaces," *Proc. Siggraph*, 1992, pp. 55–64.
4. Y. Miao, R. Pajarola, and J. Feng, "Curvature-Aware Adaptive Re-sampling for Point-Sampled Geometry," *Computer-Aided Design*, vol. 41, no. 6, 2009, pp. 395–403.
5. S. Fuhrmann et al., "Direct Resampling for Isotropic Surface Remeshing," *Proc. 2010 Vision, Modeling, and Visualization Workshop*, Eurographics Assoc., 2010, pp. 9–16.

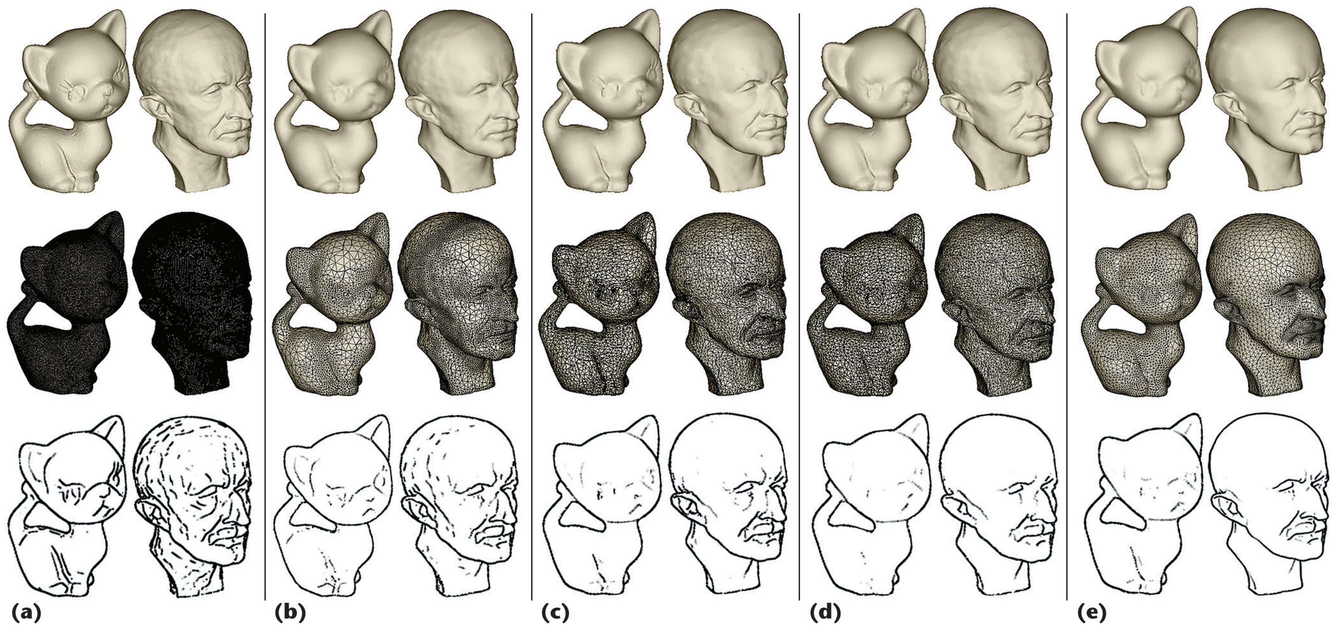

Figure 7. Comparing four remeshing approaches. We started with (a) a kitten model (approximately 137k vertices) and a Max Planck bust model (199k vertices). We remeshed them to 5k and 10k vertices, respectively, using (b) our approach, (c) a saliency-based approach,[6] (d) an approach based on quadric error metrics (QEM),[7] and (e) Simon Fuhrmann and his colleagues' approach.[8] The top row shows the shading models, the middle row shows the mesh models, and the bottom row shows line drawings generated by Doug DeCarlo and his colleagues' method.[2]

Figure 7 shows the results for remeshing a kitten model (137k vertices) and a Max Planck bust model (199k vertices). We downsampled the original models to 5k vertices (kitten) and 10k vertices (Max Planck bust) according to the visual-saliency field. Finally, we optimized and meshed the sample points. Even after we downsampled the models to less than 4 percent of their original complex-

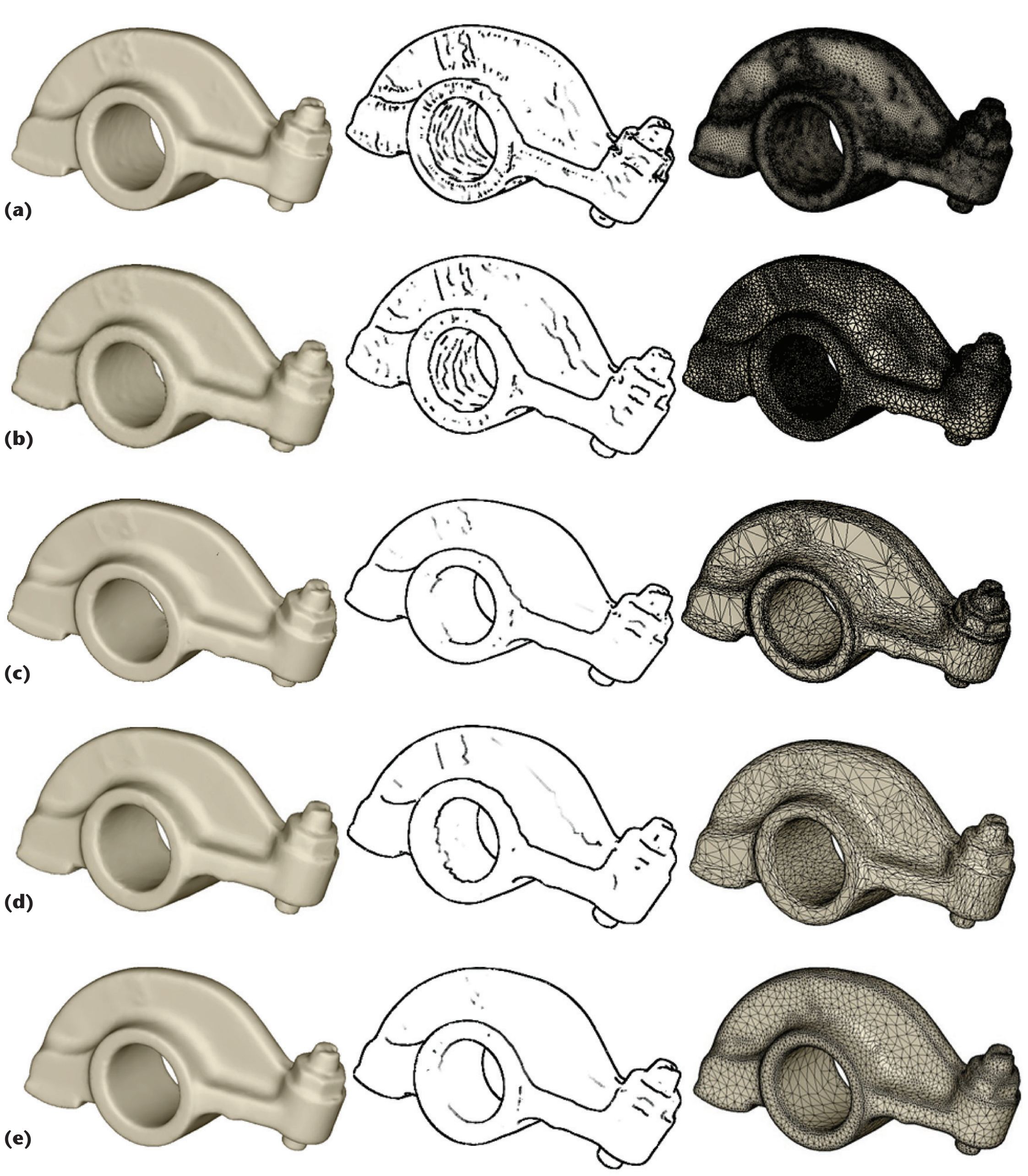

Figure 8. Remeshing a rocker arm model. We remeshed (a) the original model from 121k to 10k vertices using (b) our approach, (c) mesh saliency, (d) QEM, and (e) Fuhrmann and his colleagues' approach. The left column shows the shading models, the middle column shows the visual saliency extracted by DeCarlo and his colleagues' method, and the right column shows the mesh models. Our approach best preserved visual saliency.

ity, the resulting shapes looked similar, and the vertex distribution adapted strictly to the visual-saliency field.

We compared our approach with a saliency-based approach,[6] an approach based on quadric error metrics (QEM),[7] and Simon Fuhrmann and his colleagues' approach[8] (with a contrast factor of 1.5), using the implementation we mentioned earlier. (For more on other remeshing approaches, see the sidebar.) To compare how these approaches preserve the visual perception, we rendered the original model and remeshed models into line drawings, using a state-of-the-art method[2] in which the nonphotorealistic rendering relies on visual perception. The results in Figures 8 and 9 show that our approach best preserved visual saliency.

Using the input models' nonphotorealistic renderings as a reference, we compared the rendering results for the four approaches. We measured the quantitative similarity between the images using an image quality comparison metric simulating a human vision system.[9] Table 1 shows the results

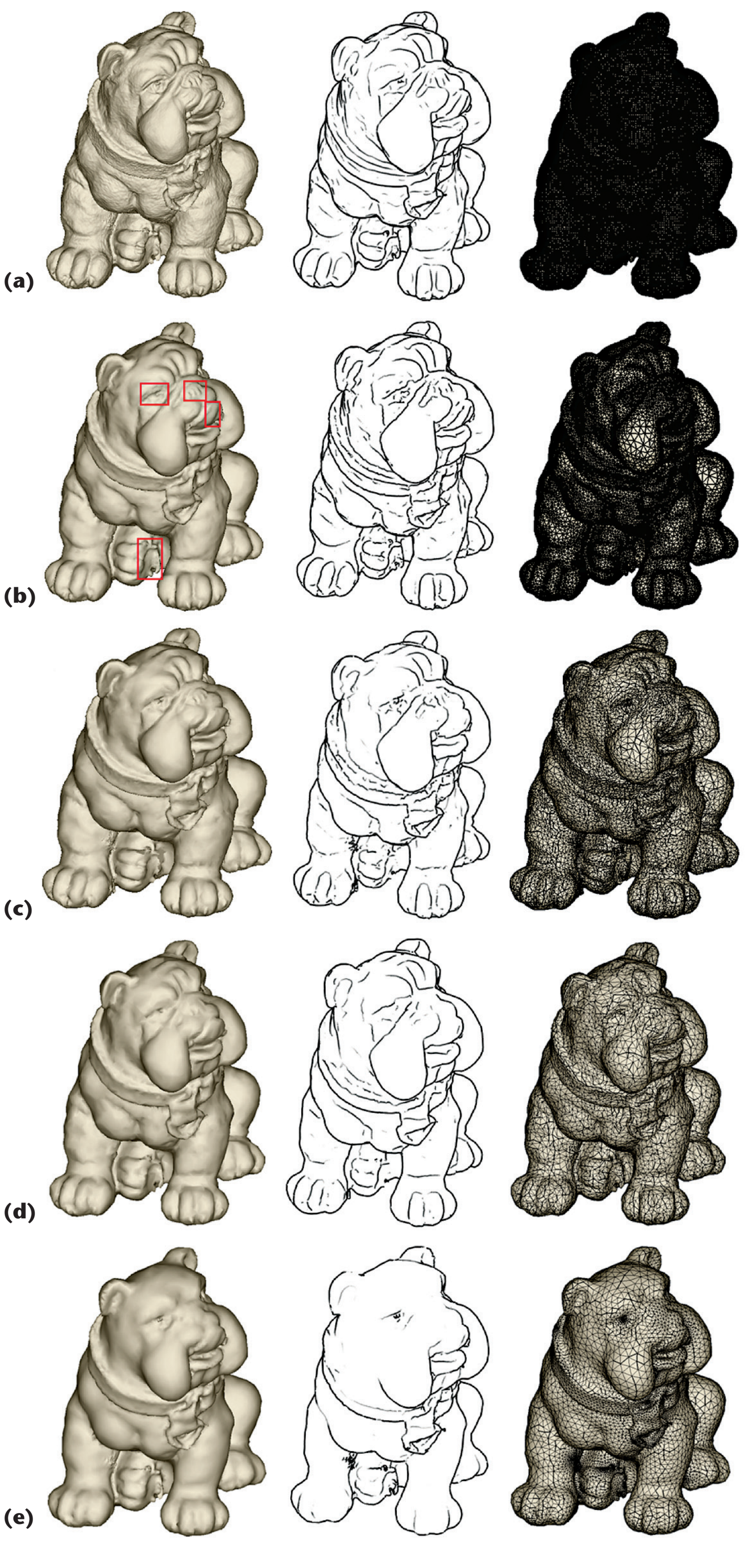

Figure 9. Remeshing a bulldog model. We remeshed (a) the original model from almost 1M triangles to 20k vertices using (b) our approach, (c) mesh saliency, (d) QEM, and (e) Fuhrmann and his colleagues' approach. The left column shows the shading models, the middle column shows the extracted line drawings, and the right column shows the mesh models. Our approach best preserved the visual saliency (see the model's eyes, nose, tongue and toe, indicated by the red boxes in Figure 9b). Moreover, our approach took only about 8 seconds with the help of the GPU's highly parallel computational power.

in the "visual error" column. A higher number indicates a model with more visual-perception features remaining. The results indicate that our approach produced the best remeshing, regarding visual saliency.

Regarding the computation times listed in Table 1, our approach was the fastest by far for input models with many vertices. This was because many steps of our remeshing pipeline are highly parallelized and run on a GPU.

We obtained the geometric error listed in Table 1 with the publicly available Metro tool (http://vcg.isti.cnr.it/activities/surfacegrevis/simplification/metro.html). The results weren't consistent with those for the visual error. Our approach's mean and maximum errors were similar but were greater than those for mesh saliency and QEM. This suggests that for applications focusing on rendering quality, existing geometric-error metrics might not fully represent visual cues.

As we mentioned before, our approach works with meshes with topology problems; see the zoomed-in views of the two polygon soup models in Figure 10. This is because our algorithms don't rely strongly on the input models' local connectivity. Figure 10 also shows the remeshing results for the two models.

Because our approach extracts the visual-saliency measurement in the 2D image space, the selection of snapshots will influence the measurement. As we mentioned before, we use six orthogonal views to generate the snapshots. So, this process might miss important features that aren't visible in those views. In the current implementation, users select the model's orientation in the front view, then our approach automatically generates the other five views. We plan to automate orientation selection by using visual cues. We'll also explore techniques for adaptive viewpoint selection to further improve the remeshed models' fidelity.

Finally, our framework isn't limited by the particular type of perception cue. So, any other extraction algorithms based on visual-perception cues can be easily plugged into it.

Acknowledgments

Hong Kong Special Administrative Region Research Grants Council / General Research Fund grant CUHK/417109 and Chinese University of Hong Kong Direct Research grants CUHK/2050518 and CUHK/2050492 supported this research. The

Table 1. Results for four remeshing approaches.

Model	No. of vertices		Approach	Time (sec.)*	Visual-similarity error[9]	Geometric error†	
	Input	Result				Max.	Mean
Kitten (Fig. 7)	137k	5k	Ours	3.635	0.9247	0.834	3.56×10^{-2}
			Mesh saliency[6]	3,882 (×1,068)	0.9110	0.178	1.78×10^{-2}
			Quadric error metrics (QEM)[7]	9.752 (×2.7)	0.8984	0.189	1.44×10^{-2}
			Fuhrmann and colleagues[8]	126.800 (×35)	0.9048	3.140	2.640
Max Planck bust (Fig. 7)	199k	10k	Ours	5.384	0.9272	1.250	0.158
			Mesh saliency	11,740 (×2,180)	0.9070	0.338	3.13×10^{-2}
			QEM	112 (×21)	0.8936	0.640	2.77×10^{-2}
			Fuhrmann and colleagues	7,403 (×1,375)	0.9021	1.750	1.360
Rocker arm (Fig. 8)	120k	10k	Ours	4.148	0.9536	2.80×10^{-3}	1.66×10^{-4}
			Mesh saliency	4,539 (×1,094)	0.9401	9.05×10^{-4}	7.80×10^{-5}
			QEM	9.361 (×2.3)	0.9399	9.19×10^{-4}	6.10×10^{-5}
			Fuhrmann and colleagues	26.210 (×6.3)	0.9367	3.75×10^{-3}	2.77×10^{-4}
Bulldog (Fig. 9)	493k	20k	Ours	8.578	0.9655	1.370	0.143
			Mesh saliency	104,781 (×12,215)	0.9535	0.602	4.03×10^{-2}
			QEM	301 (×35)	0.9573	0.878	3.79×10^{-2}
			Fuhrmann and colleagues	654 (×76)	0.9334	11.500	6.940
Horse (Fig. 10)	203k	10k	Ours	4.526	—	2.58×10^{-3}	9.10×10^{-5}
Igea (Fig. 10)	806k	6k	Ours	5.755	—	9.33×10^{-4}	3.90×10^{-5}

** The numbers in parentheses show our approach's speedup compared to mesh saliency, QEM, and Simon Fuhrmann and his colleagues' approach.*
† To measure the geometric error, we used the publicly available Metro tool.

horse polygon soup model is courtesy of Tao Ju. We downloaded the kitten, rocker arm, Max Planck, and bulldog models from the AIM@SHAPE shape repository.

References

1. J.T. Todd, "The Visual Perception of 3D Shape," *Trends in Cognitive Sciences*, vol. 8, no. 3, 2004, pp. 115–121.
2. D. DeCarlo et al., "Suggestive Contours for Conveying Shape," *ACM Trans. Graphics*, vol. 22, no. 3, 2003, pp. 848–855.
3. F. Cole et al., "How Well Do Line Drawings Depict Shape?," *ACM Trans. Graphics*, vol. 28, no. 3, 2009, article 28.
4. W. Hörmann, J. Leydold, and G. Derflinger, *Automatic Nonuniform Random Variate Generation*, Springer, 2004.
5. H. Huang et al., "Consolidation of Unorganized Point Clouds for Surface Reconstruction," *ACM Trans. Graphics*, vol. 28, no. 5, 2009, article 176.
6. C.H. Lee, A. Varshney, and D.W. Jacobs, "Mesh Saliency," *ACM Trans. Graphics*, vol. 24, no. 3, 2005, pp. 659–666.
7. M. Garland and P.S. Heckbert, "Surface Simplification Using Quadric Error Metrics," *Proc. Siggraph*, 1997, pp. 209–216.
8. S. Fuhrmann et al., "Direct Resampling for Isotropic Surface Remeshing," *Proc. 2010 Vision, Modeling, and Visualization Workshop*, 2010, pp. 9–16.
9. Y.-J. Liu et al., "Image Retargeting Quality Assessment," *Computer Graphics Forum*, vol. 30, no. 2, 2011, pp. 583–592.

***Lianping Xing** is a PhD candidate in the Chinese University of Hong Kong's Department of Mechanical and Automation Engineering. Her research interests include computer graphics, CAD, and geometric processing. Xing received an MS in computer science from Tianjin University. Contact her at lpxing@mae.cuhk.edu.hk.*

***Xiaoting Zhang** is a PhD student in the Chinese University of Hong Kong's Department of Mechanical and Automation Engineering. Her research interests include geometric modeling, image processing, and computer vision. Zhang received an MPhil in mechanical engineering and automation from the Harbin Institute of Technology. Contact her at xtzhang@mae.cuhk.edu.hk.*

***Charlie C.L. Wang** is an associate professor in the Chinese University of Hong Kong's Department of Mechanical and Automation Engineering. His research interests are geometric modeling, design and manufacturing, and computational physics. Wang received a PhD in mechanical engineering from the Hong Kong University of Science and Technology.*

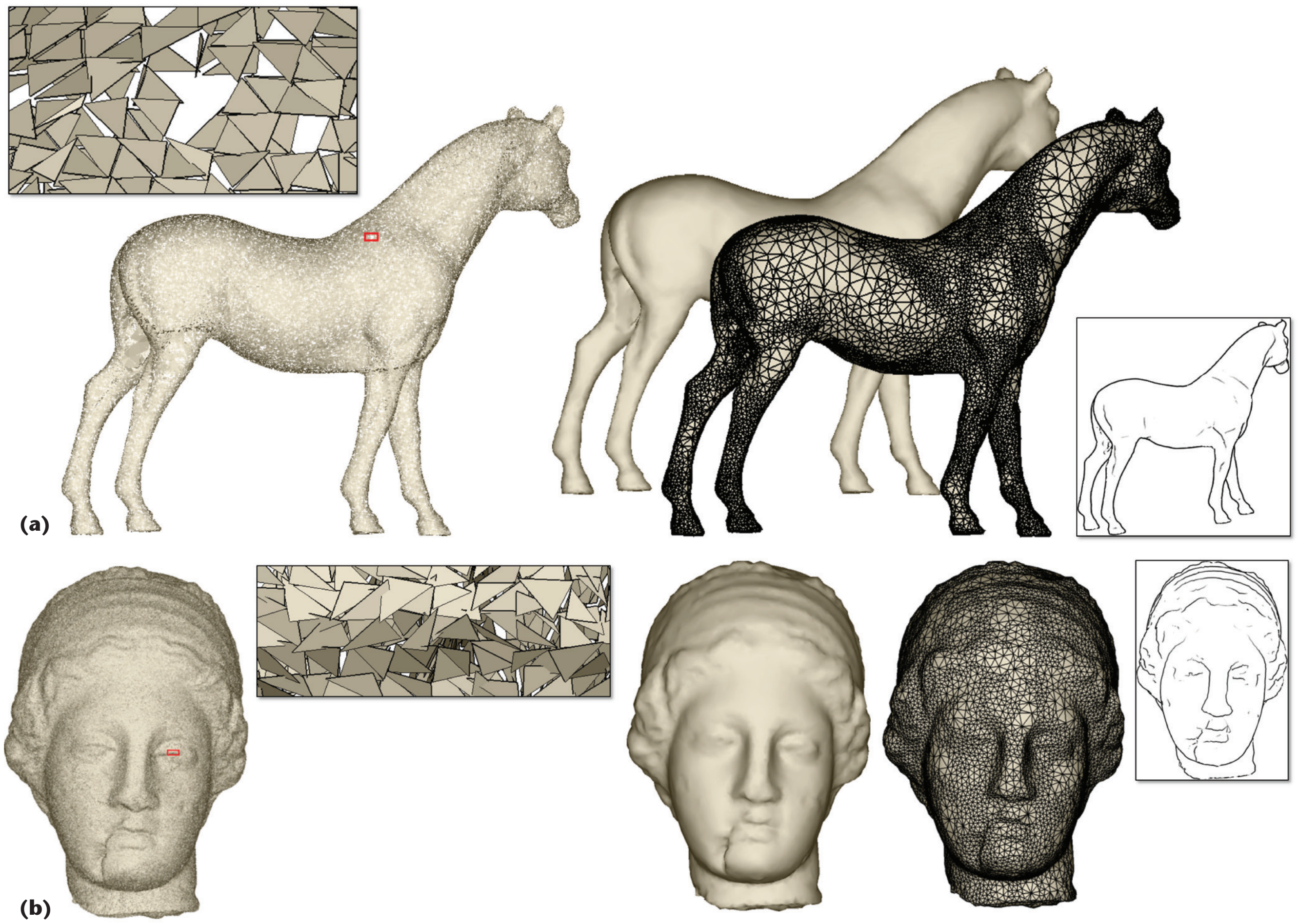

Figure 10. Remeshing two polygon soup models with topology problems (see the zoomed-in views). (a) We remeshed a horse model with 204k vertices into a two-manifold triangular mesh with 10k vertices. (b) We remeshed an Igea model with 806k vertices into a mesh with 10k vertices. In both cases, we preserved the visual saliency.

He's a fellow of ASME. He serves on the editorial boards of Computer-Aided Design, *the* ASME Journal of Computing and Information Science in Engineering, *and the* International Journal of Precision Engineering and Manufacturing. *Contact him at cwang@mae.cuhk.edu.hk.*

Kin-Chuen Hui *is a professor in the Chinese University of Hong Kong's Department of Mechanical and Automation Engineering and directs the Computer-Aided Design Laboratory. His research interests are computer graphics, geometric and solid modeling, VR, and the application of VR to environmental protection. Hui received a PhD in mechanical engineering from the University of Hong Kong. He's an editorial board member of the* Journal of Computer-Aided Design. *He's a member of the Institution of Mechanical Engineers, British Computer Society, and Hong Kong Computer Society and a fellow of the Hong Kong Institution of Engineers. Contact him at kchui@mae.cuhk.edu.hk.*