Colon Flattening with Discrete Ricci Flow

Feng Qiu, Zhe Fan, Xiaotian Yin, Arie Kaufman, and Xianfeng David Gu

Center for Visual Computing and Department of Computer Science,
Stony Brook University, New York, 11794, USA.
{qfeng,fzhe,xyin,ari,gu}@cs.sunysb.edu

Abstract. We present a novel colon flattening algorithm using the discrete Ricci flow. The discrete Ricci flow is a powerful tool for designing Riemannian metrics on surfaces with arbitrary topologies by user-defined Gaussian curvatures. Moreover, the discrete Ricci flow deforms the Riemannian metric on the surface conformally and minimizes the global distortion, which means the local shape is well preserved. Two numerical methods, the gradient descending method and Newton's method, for computing the discrete Ricci flow have been implemented. Both methods are accelerated with CUDA on the GPU. The flattened 2D rectangular mesh of the colon is rendered using volumetric ray-casting method with pseudo color to produce electronic biopsy images.

Key words: Colon flattening, discrete Ricci flow, conformal map, GPU

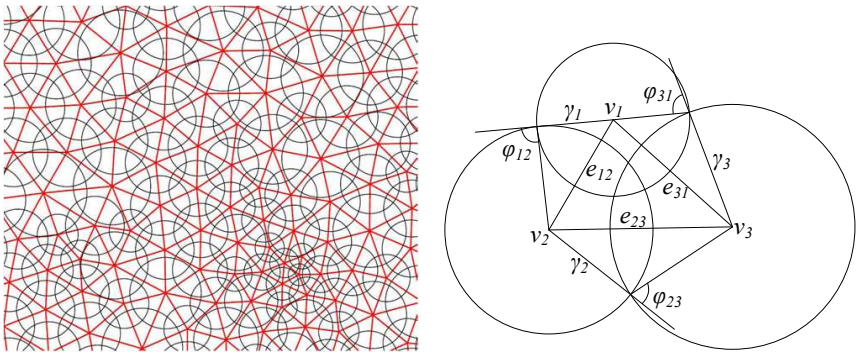
1 Introduction

Virtual colonoscopy has been proven to be an efficient and convenient method of detecting colon polyps, the precursor of cancer. Traditionally, virtual colonoscopy uses a virtual fly-through visualization system for the physician to navigate inside the virtual colon model. However, due to the length and structure of the colon, inspecting the entire colon wall is time consuming and error prone. Moreover, polyps behind folds may be hidden, which results in incomplete examinations.

Virtual colon flattening is an efficient visualization technique for polyp detection, in which the entire inner surface of the colon is dissected and flattened on a 2D plane. However, virtual colon flattening introduces distortion. The existing colon flattening methods can be divided into two major categories: area preserving methods [1] and angle preserving methods [2, 3]. The angle preserving methods preserves the local shape on the flattened colon surface, which is desirable because the colon polyps usually have a semi-ellipsoidal shape which we want to preserve. Moreover, the flattened colon can be efficiently volume rendered to produce the electronic biopsy image for computer aided polyp detection [7]. We propose a new colon surface flattening that is angle preserving based on the discrete Ricci flow method. The discrete surface Ricci flow [4] has a simple physical intuition, is general for surfaces with arbitrary topology, and can be efficiently accelerated with the graphics processing unit (GPU).

2 Discrete Ricci Flow

Ricci flow has a simple physical intuition. Given a surface S with a Riemannian metric $\mathbf{g}$, the metric induces the Gaussian curvature function. If the metric $\mathbf{g}$ is changed, then the Gaussian curvature will be changed accordingly. The Ricci flow deforms $\mathbf{g}$ in the following way: at each point, $\mathbf{g}$ is locally scaled to a new metric $\bar{\mathbf{g}}$ such that the scaling factor is proportional to the curvature at the point. Because of this locally isotropic deformation, the deformation is a *conformal* metric deformation such that angles measured by $\mathbf{g}$ are equal to that measured by $\bar{\mathbf{g}}$. After the deformation, the new metric $\bar{\mathbf{g}}$ induces a new curvature function. Both the metric and the curvature evolve while the deformation process is repeated. And the curvature evolution is like a heat diffusion process. Eventually, the Gaussian curvature function is constant everywhere. For a surface with a cylinder topology such as the colon, the result Gaussian curvature function is 0 everywhere.



(a) (b)

Fig. 1. (a) Flat circle packing metric, and (b) circle packing metric on a triangle.

In practice, the colon surface is represented with a triangular mesh. Therefore, the discrete Ricci flow is computed where the Riemannian metric is replaced with the circle packing metric. As shown in Fig. 1, let Γ be a function that assigns a radius γ_i to vertex v_i; let Φ be a function that assigns an acute angle ϕ_{ij} to half edge e_{ij} as the edge weight. The pair of vertex radius function and edge weight function (Γ, Φ) defined on a mesh Σ is called the circle packing metric of Σ. The discrete Ricci flow is defined as:

$$\frac{du_i(t)}{dt} = (\bar{K}_i - K_i) \tag{1}$$

where $u_i = \log \gamma_i$ and $\bar{K}_i$ is the target Gaussian curvature. For colon surface flattening, the target Gaussian curvature is 0 everywhere. The discrete Ricci flow deforms the circle packing metric according to the discrete Gaussian curvature,

just like the Ricci flow deforms the Riemannian metric according to the Gaussian curvature.

The pipeline of the discrete Ricci flow based colon flattening is as follows:

1. Compute the initial circle packing metric of the colon surface mesh;
2. Deform the circle packing metric with Eqn. 1;
3. Compute the layout of the flattened mesh with the result metric.

To compute the initial circle packing metric, the radii of vertices are computed with:

$$\gamma_i^{jk} = \frac{1}{2}(l_{ki} + l_{ij} - l_{jk}) \tag{2}$$

$$\gamma_i = \frac{1}{m} \sum_{f_{ijk} \in F} \gamma_i^{jk} \tag{3}$$

where l_{ij}, l_{jk}, and l_{ki} are the edge length of triangle f_{ijk}. Then, the half edge weight (or the angle associated with each half edge) ϕ_{ij} can be computed with the cosine law in Euclidean space.

To deform the circle packing metric, one method is the gradient descending algorithm in which the discrete Gaussian curvature of every vertex is computed to update u_i

$$u_i = u_i + \epsilon(\bar{K}_i - K_i) \tag{4}$$

where ϵ is a small constant (step size). Then, u_i is used to calculate the vertex radii of the updated circle packing metric. The discrete Gaussian curvature can then be deduced from the updated circle packing metric. This process repeats until the maximum curvature error is less than certain threshold. Another method is the Newton's method where the sparse Hessian matrix $H = (\partial K_i / \partial u_j)_{n \times n}$ is computed in each step. Then the linear system:

$$H d\mathbf{u} = \bar{\mathbf{K}} - \mathbf{K} \tag{5}$$

is solved to update the vector $\mathbf{u} = (u_0, u_1, \cdots, u_{n-1})^T$.

Finally, the flattened mesh layout is computed by embedding the triangles with edge lengthes in the resulting circle packing metric. We refer the reader to [4] for more details.

3 GPU Acceleration and Results

The GPU is the processor on a commodity 3D graphics card for accelerating raster-based rendering. In recent years, the GPU performance has been increasing at incredible rate. Current GPU surpasses CPU in raw computational power by an order of magnitude and the gap between GPU performance and CPU performance is still increasing. The high performance and fast performance growth of the GPU is made possible by the explicit parallelism in raster-based rendering. Because the GPU is data driven (i.e., it needs to process a huge amount

of vertices, geometry primitives, and pixels in real-time speed), it emphasizes data parallelism. For example, the NVIDIA GeForce 8800 GTX has 128 thread processors that are dynamically allocated to data processing and compute simultaneously. The NVIDIA GeForce GTX 280 released this year has 240 thread processors for data parallel computation. In addition to its high computational power, the GPU is becoming more and more flexible and programmable. High level languages have been added to graphics APIs, such as OpenGL and DirectX, for programming the GPU. As a result, general-purpose computation on the GPU (GPGPU) has become an active area of research. A wide range of general-purpose application has been accelerated on the GPU.

Compute Unified Device Architecture (CUDA) is a new programming tool developed by NVIDIA for general-purpose computation. With CUDA, the programmer can use extended C language to develop general-purpose computation on the GPU. No knowledge in graphics hardware and APIs is needed anymore. In addition, CUDA allows the program to access a linear GPU memory, which is larger and much more convenient to use than the GPU textures. CUDA further exposes to the programmer hardware features, such as data parallel cache and scatter operation.

To implement the gradient descending algorithm with CUDA, the surface mesh is stored as two arrays, vertex array and triangle array. The vertex array stores pre-vertex data, including vertex radius, target curvature, boundary flag, incident triangle list, and incident corner angles. The triangle array stores per-triangle data, including indices of vertices, half edge weights, and corner angles. Two computation kernels operate on the arrays in a SIMD fashion. The first computation kernel works on the triangles and calculates edge length and corner angles out of vertex radii. Based on corner angles, the second computation kernel updates the curvature for every vertex. This process repeats until the maximum curvature error is below a user defined threshold. Because each computation kernel execute on an array (triangle or vertex) and every data element in the array is processed independently, the computation is fully parallelized.

For the Newton's method, the major difference from the gradient descending algorithm is updating the Hessian matrix $(h_{ij})_{n\times n}$ and solving the linear system in Eqn. 5. The elements of the Hessian matrix $h_{ij} = \partial K_i/\partial u_j$ are calculated with a computation kernel in parallel. The core of the linear system solver is the multiplication of a sparse matrix and a dense vector. Bolz et al. [5] have proposed a scheme of sparse matrix solver using OpenGL, where the off-diagonal entries of the sparse matrix are compactly stored in row major. To efficiently implement the sparse matrix solver with CUDA, we propose a new method to store the sparse matrix compactly in a 1D array. First, the rows of the matrix are sorted in descending order by the number of entries n_i. Denote the reordered row as $R_{i'}$. Second, each row $R_{i'}$ is condensed by removing the zero entries, and the non-zero entries are associated with the corresponding column indices. Denote the j-th non-zero entry in row $R_{i'}$ as $h_{i'j}$. Then, all $h_{i'j}$ are stored linearly in column major, which means i' varies first. With this column major storage, the memory

access in CUDA implementation can be efficiently coalesced. Other operations such as vector dot product are implemented with the CUBLAS library [6].

We have tested our GPU accelerated colon flattening algorithm with colon datasets from NIH. The dataset has been preprocessed with the method in [7] for digital cleansing, segmentation and colon surface mesh extraction. Fig. 2 shows the electronic biopsy image of a flattened colon rendered with volumetric ray casting method using a transfer function with pseudo color. The experiment is conducted on a PC platform with dual Xeon 3.6GHz CPU and an NVIDIA GeForce 8800 GTX graphics card. In our experiment, the GPU implementation of the gradient descending algorithm is approximately 40 times faster than on the CPU. The Newton's method on the GPU is approximately 8 times faster than the software implementation.

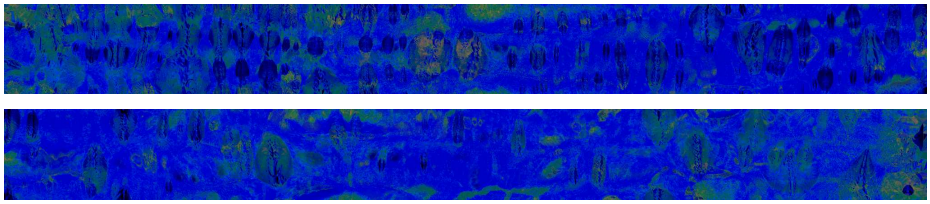


Fig. 2. Electronic biopsy image of a flattened colon. Red and yellow color represent high densities while low density regions are blue.

4 Conclusions

We have presented an angle preserving colon flattening method based on the discrete Ricci flow computation, which has an advantage of being fully parallelizable. We have implemented the algorithm on a GPU with the NVIDIA CUDA programming toolkit. The GPU implementation achieves a speedup factor of 40 for the gradient descending method and 8 for the Newton's method. As GPUs continue to become more powerful and programmable, their impact on medical imaging will increase.

References

1. Bartrolí, A.V., Wegenkittl, R., König, A., Gröller, E.: Nonlinear virtual colon unfolding. IEEE Visualization (2001) 411–420
2. Haker, S., Angenent, S., Tannenbaum, A., Kikinis, R.: Nondistorting flattening maps and the 3D visualization of colon CT images. IEEE Transactions on Medical Imaging **19**(7) (2000) 665–670
3. Hong, W., Gu, X., Qiu, F., Jin, M., Kaufman, A.: Conformal virtual colon flattening. Symposium on Solid and Physical Modeling (2006) 85–93

4. Jin, M., Kim, J., Luo, F., Gu, X.D.: Discrete surface ricci flow. IEEE Transaction on Visualization and Computer Graphics **14**(5) (2008) (in press)
5. Bolz, J., Farmer, I., Grinspun, E., Schröoder, P.: Sparse matrix solvers on the GPU: conjugate gradients and multigrid. ACM Transactions on Graphics **22**(3) (2003) 917–924
6. NVidia: Compute Unified Device Architecture: Programming Guide. (2007) Version 1.1, http://www.nvidia.com/object/cuda_develop.html.
7. Hong, W., Qiu, F., Kaufman, A.: A pipeline for computer aided polyp detection. IEEE Transactions on Visualization and Computer Graphics **12**(5) (2006) 861–868