An Optimization of Square Parameterization

Anuwat Dechvijankit Hiroshi Nagahashi Kota Aoki
Department of Computational Intelligence and Systems Science, Tokyo Institute of Technology.
4259 Nagatsuta-cho, Midori-ku, Yokohama-shi,
Kanagawa, Japan, 226-8503
dechvijankit.a.aa@m.titech.ac.jp longb@isl.titech.ac.jp aoki.k.af@m.titech.ac.jp

ABSTRACT

In order to parameterize a three-dimensional surface into a two-dimensional planar domain, we need to convert its polygonal mesh into a disk topology surface. For quality of texturing or re-meshing that uses a parameterization technique, it is more effective if the distortion of two-dimensional manifold planar domain map is as small as possible. Since square planar domain is easily understandable to human or very simple as a computer image file, it has been frequently used in real world applications. We introduce a series of experiments focusing on how to deliver an optimized square parameterization with low-cost calculation and stable result. The result of these experiments shows that our method is a suitable method for optimizing square parameterization.

Keywords

Mesh Parameterization, Optimization, Particle Swarm Optimization, Sampling

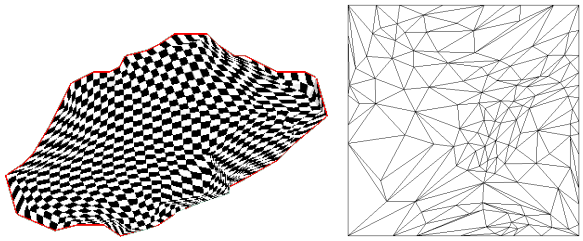
1 INTRODUCTION

Mesh parameterization is defined as a mapping between a 3D manifold surface and a suitable target domain. In general, the mesh parameterization is formulated as a mapping from 3D triangulated surfaces to a certain 2D planar domain. However, it requires the surfaces to be topologically equivalent to a disk without any hole. Parameterization between two domains generally causes distortion errors such as a stretch. Hence, low stretching is an important criterion for parameterization.

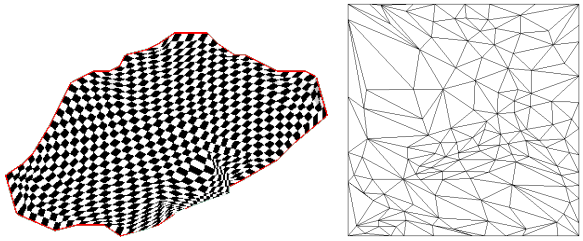
Unlike a circular boundary domain or a natural boundary domain, the square boundary domain has a specific characteristic that requires user-defined algorithms in boundary-mapping assignment (constraint part). Different positions in square boundary mapping can generate different quality of parameterization result as shown in figure 1.

The easiest way for delivering the lowest stretching square parameterization is to check all possible boundary mappings (brute-force) with a stretch-minimizing parameterization method. It can guarantee the best result, however the main problem of this approach is extensively time-consuming.

We did a series of experiments getting the lowest distortion square parameterization by avoiding the brute-force high-computation method. The main goal of this paper is to devise an algorithm to deliver an optimal square parameterization from any kind of stretch-minimizing methods with fast and stable result. Moreover, since GPU-Computing has been introduced and widely been used nowadays, the devised algorithm should support parallel-computing scheme as well.



(a) L^2 stretch: 1.320295



(b) L^2 stretch: 1.175196

Figure 1: Ustica model with check-board texture mapping using same stretch-minimizing square parameterization with different boundary mappings. (a) shows the worst case that have largest L^2 stretch. (b) shows the best case that have smallest L^2 stretch. We can notice the different quality of textures around boundary area.

 ISBN 978-80-86943-75-6

2 RELATED WORKS

Since texture mapping was introduced into computer graphics world by E. Catmull [Coo87a], it became a trend that every graphics board or graphics API must support it nowadays. Mapping a 2D texture onto a 3D surface requires some kind of parameterization of the surface. The parameterization result can be represented by planar coordinates u and v, indicating a position in 2D domain related to a position (x,y,z) in 3D domain.

Many well-known parameterization methods have been proposed. Tutte[Tut63a] used a barycentric mapping theory and created a conformal mapping. Floater[Flo97a] used relative angles as a weight in each interior vertex to create barycentric mapping. Later on, stretch-minimizing methods have been proposed to achieve low stretch as possible. Sander et al. [San01a] used geometric-stretch matrix as the sum of squared singular values, and minimized it as a non-linear system. Yoshizawa[Yos04a] proposed a fast method of stretch-minimizing by recomputing the weight of linear energy-minimizing equations by using previous stretch value as a divisor.

Concerning the limitation of fixed-boundary parameterization, Least Squares Conformal Maps (LSCM)[Lev02a] were presented as alternative ways to optimize the boundary positions from fixed-boundary into free-boundary. They used different harmonic energy formulations found in harmonic map[Eck95a] but still minimized angular distortion. Intrinsic parameterizations[Des02a] used the same technique found in LSCM to preserve angle distortion, and preserved area distortion. Both of them could significantly improve the distortions. However, they aimed to optimize by changing fixed-boundary into free-boundary parameterization, not square-boundary one.

To guarantee the generation of a valid parameterization without local or global fold overs and the control of each mesh triangle distortion to not exceed a certain threshold, Sorkine[Sor02a] proposed bounded-distortion concept with simultaneously seam-cutting. Lipman[Lip12a] also proposed bounded-distortion mapping spaces which can control worst-case conformal distortion, orientation preserving and one-one mapping in various existing mapping algorithms. However, they aimed to control mappings at unconstrained part.

3 SQUARE BOUNDARY MAPPING

Generally, planar parameterization requires some constraint values in its solving system. For doing a fixed-boundary parameterization, it requires a user-defined boundary position mapping as constraint values before solving interior coordination. Unlike circular parameterization that averages each boundary edge length and circle angle, square parameterization needs some user-defined algorithms in the assignment of boundary mapping. There are no specific algorithms for it. Users can create their own algorithm based on the length or numbers of boundary edges and so on.

In our square boundary mapping algorithm, a mesh M has n boundary vertices. Let boundary edges be $E_B = ((v_{b^1}, v_{b^2}), (v_{b^2}, v_{b^3}), \ldots, (v_{b^n}, v_{b^1}))$, having total length d. Let a list of all boundary vertices be $V_B = (v_{b^1}, v_{b^2}, ..., v_{b^n})$ and it is sorted in the order of connection of E_B. We call v_{b^1} as a reference start point of V_B and E_B. Let $P_{i,j}$ be a corner position in square planar domain as shown in figure 2 and u_s be the length of each side of square planar. We try to map some edges in E_B onto a side of square planar. In order to achieve lowest stretch at boundary area, one side should be mapped by a quarter of E_B based on the total length of edges ($0.25d$).

Let v_b be a vertex in V_B that we want to map onto $P_{0,0}$. Let the distance from v_b to v_{b+m} be l. We try to find v_{b+m} whose distance l is equal $0.25d$. However, in most cases it is not equal. Therefore, we find v_{b+m} that has $l \approx 0.25d$. Then, we map $(v_b, v_{b+1}, \ldots, v_{b+m})$ whose total length is l onto the square side from $P_{0,0}$ to $P_{1,0}$ relatively on each edge length over u_s.

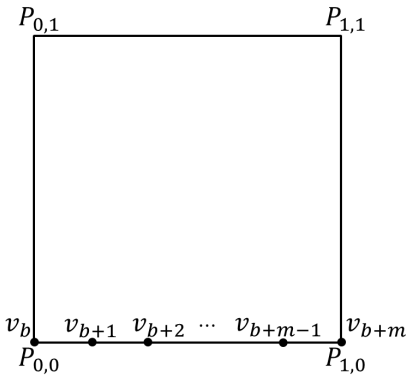


Figure 2: shows a mapping sequence on planar points from $P_{0,0}$ to $P_{1,0}$.

As for other sides of the square, we can follow the same process described above by locating v_{b+m} to the corner $P_{1,0}$ and iterate the mapping process to $P_{1,1}$, $P_{0,1}$ and finally back to $P_{0,0}$.

4 OPTIMIZATION

Square parameterization has a unique characteristic that requires a user-intervening boundary position mapping. The different boundary positions in planar domain can give moderate margin of stretch (see figure 1). The problem is how we can obtain the optimal square parameterization.

Considering our problem of square boundary optimization, we map the boundary vertices in mesh domain

 ISBN 978-80-86943-75-6

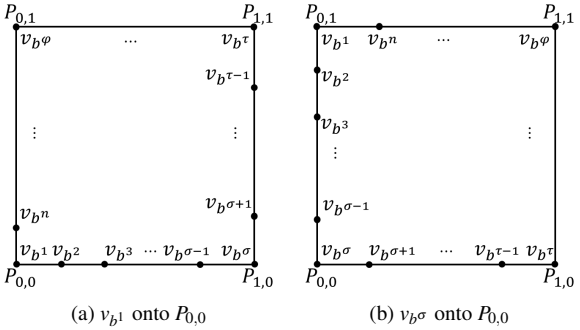
onto the boundary in square domain with some conditions. One important condition is the relationship between a side of square boundary and mapping boundary edges. Our mapping method is trying to map a quarter of total boundary edges in terms of length onto one side of the square that we mentioned in section 3. With this condition, a new complexity arises. That is, each boundary mapping could have different number of edges on each side of the square. It is impossible to incorporate these boundary conditions into a linear solving system.

The simplest way is a brute-force approach using a stretch-minimizing parameterization. In our mapping method, brute-force means that we let every vertex in V_B be mapped onto $P_{0,0}$ then do stretch-minimizing parameterization. Although it guarantees the best answer, one time stretch-minimizing parameterization on a fine mesh might consume not a few amount of time. Although, we can speed up by applying parallel-processing but doing brute-force and checking every possible boundary mapping on the square boundary might consume a lot of time.

25 percent of brute-force

Let the first boundary mapping be v_{b^1} onto $P_{0,0}$, v_{b^σ} onto $P_{1,0}$ and v_{b^τ} onto $P_{1,1}$ ($v_{b^1}, v_{b^\sigma}, v_{b^\tau} \in V_B$) that are assigned by our boundary mapping algorithm. It means the distance from v_{b^1} to v_{b^σ} should be approximately $0.25d$, also the same for distance from v_{b^σ} to v_{b^τ}.

By starting from v_{b^1}, we sequentially assign a vertex v_b onto $P_{0,0}$, and map the following vertices onto the square domain $[P_{0,0}, P_{1,0}]$. After repeating the mapping for interval of a quarter of boundary edges, then the vertices v_{b^σ} and v_{b^τ} might be mapped onto $P_{0,0}$ and $P_{1,0}$ respectively. It is the same as we rotate the first mapping (v_{b^1} onto $P_{0,0}$) 90 degree as shown in figure 3.



(a) v_{b^1} onto $P_{0,0}$ (b) v_{b^σ} onto $P_{0,0}$

Figure 3: shows a similarity of boundary mapping after shifting for a quarter of total length of boundary edges.

From this property, we can reduce the number of testing cases to around 25 percent because we can rotate the parameterized planar from a boundary mapping $(v_{b^1}, v_{b^2}, \ldots, v_{b^\sigma})$ to obtain the result of the rest mapping $(v_{b^{\sigma+1}}, v_{b^{\sigma+2}}, \ldots)$. However, doing brute-force with around 25 percent of total testing cases still consumes much of time. Our goal is finding the optimal square parameterization while keeping the calculation time less than 25 percent of brute-force.

4.1 Faster parameterization methods

We examined the time consuming issue of the brute-force approach. We could notice that stretch-minimizing parameterization is a main reason of the problem. Since it is iterative process and it seeks converging of the energy or stretch, its calculation time is much more than the time of one parameterization by means of linear solving system. If we can replace a stretch-minimizing to a faster solving parameterization in the optimization process, then it should reduce the consuming time a lot. Here, we set our hypothesis that every parameterization method should give a same direction result; the best boundary mapping by fast-solving method is same as the boundary mapping by stretch-minimizing method. Now, the concept "same direction" is defined as follow:

Same Direction

We assume that there are two square boundary mappings; M_n and M_{n+1}. In addition, we have two parameterization methods F and G. Let R_n and R_{n+1} be parameterization results if we do F on both M_n and M_{n+1}. Moreover, S_n and S_{n+1} are parameterization results if we do G on both M_n and M_{n+1}. If R_n is better than R_{n+1} and S_n is better than S_{n+1} then we say that F and G have a same direction.

4.1.1 Experiment and Result

We setup the experiment to test the hypothesis. One parameterization is fast solving but high potential of stretch. The other is slow solving but stretch-minimizing. We tried to use fast solving methods to predict which boundary mapping is the best for square parameterization. We did the experiment to observe the stretch of unit square boundary by various fast solving parameterization methods of Shape-Preserving[Flo97a], Tutte[Tut63a] , mean-value[Flo03a] and harmonic map[Eck95a]. We compare L^2 stretch[San01a] value from our candidate methods with value from stretch-minimizing square parameterization using [Yos04a].

As a result, we could not find any relationship among them. When observing the lowest stretch of all methods with same boundary positions, they do not give the lowest stretch in the same direction. Stretch observation from the fast solving method does not enable to check which mapping boundary points is optimal setting for stretch-minimizing method as the hypothesis.

We concluded that optimizing square parameterization needs to be done and analyzed directly from the stretch-minimizing method.

 ISBN 978-80-86943-75-6

4.2 Heuristics

From previous experimental result, we cannot use a fast solving parameterization. Instead, we need to use stretch-minimizing one that requires a lot of calculation. For that reason, we set our approach reducing the number of calculations. If we can pursue the optimization by doing parameterization with few testing cases, then it will surely be better than brute-force method.

We attempted to find a best optimization among the existing ones. There are many techniques and they are divided into 3 categories[Wik13a]. Optimization algorithms such as a Simplex algorithm are suit for linear or quadratic programming solving. Iterative methods such as Newton and Quasi-Newton methods suit for non-linear programming solving. Heuristic algorithms such as Genetic algorithms or Hill climbing suit for solving the problems that cannot be solved or too slow by classic methods.

When we consider our mentioned problem, we concluded that a heuristic algorithm may be the best one for boundary optimization problem. The reason is that our boundary optimization problem has the difficulty of two sub-problems connected together. One sub-problem is concerned with our main problem, i.e., finding best mapping of boundary vertices and edges. The other is a parameterization problem of finding planar location of interior vertex. It is too difficult to combine the two sub-problems into one problem solving system since it has unique condition for the boundary mapping. In addition, we try to find global optimum of our problem, not local ones. To find one local optimum does not show or know it is globally optimal until all local optimums are found. From these reasons, well-known algorithms such as Simplex or Newton do not fit to our problem. Hence we chose a heuristic method to solve our optimization problem.

4.2.1 Particle Swarm Optimization

We choose "Particle Swarm Optimization" algorithm (PSO) [Ken95a] for solving our optimization problem. It is a new swarm intelligent technique, originally inspired by social behavior of animal flocking. PSO has been used mainly to solve unconstrained, single-objective optimization problems. The advantage of using PSO is that it does not use the gradient of the problem to be optimized, so the method can be readily employed for optimization problems. This is especially useful when the gradient is too laborious or even impossible to derive. This versatility comes at a price, as PSO does not always work well and may need tuning of its behavioral parameters so as to perform well on the problem at hand[Eri10a]. It requires 3 parameters of effective factor from velocity, local-best position and global-best position.

Next velocity of each particle is updated based on current velocity, local-best and global-best positions of the particle. Moreover, it is received effectively from these 3 parameters and random numbers. Then, each particle's next position is calculated by using its new velocity. Let i be the number of particles, k be iteration times and r be a random number in searching scope. Then, we assume that a, b_l and b_g are user-defined effective factor from current velocity, local-best position and global-best position, related as follows respectively. In equation forms, they are

$$v_i^{k+1} = av_i^k + b_l r_1(pl_i^k - x_i^k) + b_g r_2(pg^k - x_i^k) \quad (1)$$

$$x_i^{k+1} = x_i^k + v_i^{k+1} \quad (2)$$

Algorithm 1 summarizes a standard PSO algorithm.

```
\\initialize particle...
for each particle x_i do
    x_i ← r
    pl_i ← unknown
\\starting main iteration...
repeat
    for each particle x_i do
        y_i^k = f(x_i^k)
        if y_i^k better than f(pl_i^k) then
            pl_i^k ← x_i
    pg^k ← best of all x_i^k
    for each particle x_i do
        v_i^{k+1} ← a v_i^k + b_l r_1(pl_i^k − x_i^k) + b_g r_2(pg^k − x_i^k)
        x_i^{k+1} ← x_i^k + v_i^{k+1}
until k > maximum iterations or
      pg unchanged many times
optimum ← pg
```

Algorithm 1: Pseudo-code of PSO algorithm

4.2.2 Experiment and Result

We did experiments of applying one dimensional PSO to our problem. Let a particle position x_i be the distance from the reference start point v_{b^1}. First, search the nearest v_{b^α} in V_B whose distance from v_{b^1} is closest to x_i, and assign it to v_b that is mapped at lower-left corner $P_{0,0}$. Then we follow the algorithm described in section 3. At last, we do square stretch-minimizing parameterization using [Yos04a] method to obtain interior coordinates in 2D planar domain and then calculate L^2 stretch value for evaluation. On each mesh, different number of particles and various changing factors of local and global best positions were examined until the system gave optimal answer. We did 10 times per one parameters-setting and get statistic results since PSO algorithm is based on a random process as shown in equation 1.

As a result, we could reduce the calculation time to around 50 to 75 percent comparing to 25 percent of

 ISBN 978-80-86943-75-6

brute-force approach. By changing user-defined parameters, we could improve calculation time in some mesh models. However, we could not gain an expected result from the turning of these user-defined parameters yet.

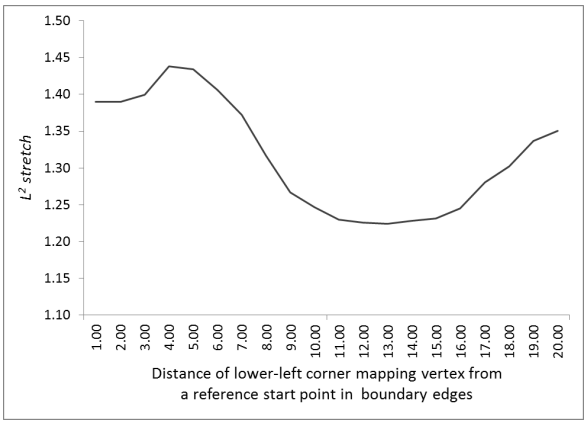
The number of particles plays important roles; more particles secure the best answer (same as brute-force's answer) but calculation cost increases. The algorithm itself is based on random process, and it may give unstable optimal answer if we use small number of particles. Increasing the number of particles will cost almost the same calculation time as 25 percent of brute-force approach. Moreover, PSO has a disadvantage point on parallel-computing because it updates particle positions based on global-best position at each iteration which prevents from doing large numbers of parallel-processing simultaneously.

We concluded that PSO can improve the performance when using an appropriate number of particles. Even, the performance was still not as we expect but its algorithm of checking few positions and focusing around seems to be appropriate with our optimization problem. The main bottleneck of PSO in our problem is a procedure of random search. Avoiding the random search while checking few positions should generate better performance stably.

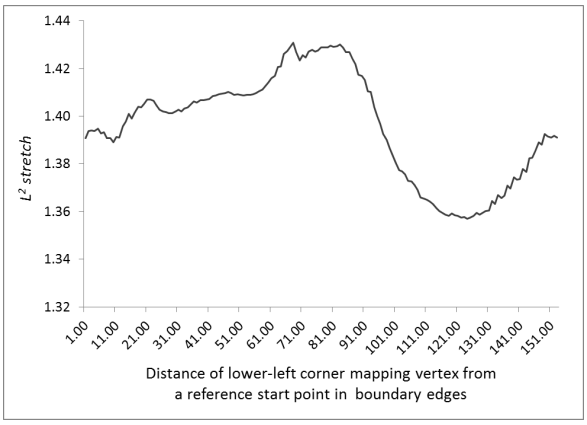
4.3 Sampling

The PSO algorithm is based on a sampling approach. Each particle acts as a sampling unit and every time a particle moves to a new position, it will examine that position. In our case, it means doing one parameterization. If we observe a particle movement, we can notice that the particle will move toward previous local-best and global-best positions. Therefore, initial global-best particle is important that affects to the performance and stability. Each particle's initialization can narrow down searching scope a lot if a position is located at optimum area because the global-best particle is one of local-best particles. We adapted these concepts into a sampling algorithm by narrowing down searching scope of the initial sampling.

We started to analyze square parameterization by looking at brute-force results. We noticed that most of our test models' stretches are changed gradually when we change v_b at $P_{0,0}$ along V_B (see figure 4). From this characteristic, we can reduce the number of calculations by focusing attention on the boundary-mappings that have high potential to give an optimal result. In order to be able to do such thing, we need to know where is appropriate searching scope. If we plot stretch values, we are searching for potential area having global optimum. In other words, it is important to find a list of mapping $(v_{b-p}, \ldots, v_b, \ldots, v_{b+q})$ at $P_{0,0}$ that generates an optimal square parameterization.



(a) hand model



(b) Stanford bunny model

Figure 4: The graphs that show stretch values from doing square stretch-minimizing parameterizations (25 percent of brute-force) on testing models.

The problem is how to determine the searching scope. We use a sampling approach as a survey of stretch values same as PSO. Sampling is the reduction of a signal. A common example is the conversion of a sound wave (a continuous signal) to a sequence of samples (a discrete-time signal). In our problem terms, we reduce the number of parameterizations to few cases so we can determine our searching scope. We do not use complex algorithms like pattern-search or random-search but we do a static sampling.

4.3.1 Step-Sampling

We propose a simple algorithm named step-sampling. It samples stretch values from selected boundary mappings. They will be selected as a step defined by user. After sampling was completed, we will get a boundary mapping that gives the lowest stretch as a center of optimal area. At last, we do deep-checking in that area. We check its neighbors that are still be unchecked for stretch values. The test case (boundary mapping) that gives the lowest stretch is an optimal answer.

Algorithm 2 summarizes our step-sampling algorithm.

 ISBN 978-80-86943-75-6

Model	Total Test Cases	step											
		2	3	4	5	6	7	8	9	10	11	12	13
Hand	18	61.11%	**44.44%**	**44.44%**	**44.44%**	**44.44%**	50.00%	55.56%	55.56%	61.11%	66.67%	72.22%	77.78%
Head	20	60.00%	**55.00%**	**55.00%**	60.00%	70.00%	70.00%	85.00%	95.00%	100.00%	100.00%	100.00%	100.00%
Bunny Holes	153	51.63%	35.95%	29.41%	25.49%	23.53%	22.22%	22.22%	**21.57%**	22.22%	22.22%	22.88%	23.53%
Bunny No hole	123	52.03%	36.59%	30.08%	23.58%	21.14%	19.51%	18.70%	**17.89%**	**17.89%**	**17.89%**	**17.89%**	**17.89%**
Max	112	51.79%	37.50%	30.36%	27.68%	25.89%	**25.00%**	**25.00%**	25.89%	26.79%	27.68%	28.57%	29.46%
Cow	240	50.83%	35.00%	27.50%	23.33%*	20.83%	19.58%*	18.33%	17.92%*	**17.50%***	**17.50%**	**17.50%**	17.92%*

Table 1: sampling method result: symbol * indicates that optimal answer is not same as brute-force's result. Bold numbers mean the lowest percentage of calculation.

$step \leftarrow 2,3,\ldots$ *user defined step*
m = number of vertices in first $0.25d$ of E_B
$stretch[\,] \leftarrow \infty$ *array m size*

initial step sampling
for $i \leftarrow 1$ to m **do**
  if $i \bmod step = 1$ **then**
    $stretch[i] \leftarrow SquareParam(v_{bi}$ as $P_{0,0})$

deep-checking
$l \leftarrow$ index of $stretch[\,]$ that has best result
$J[\,]$ =[...,l-2,l-1,l+1,l+2,...] where
  $stretch[\,] = \infty$ and close to l
for $j \leftarrow$ each J **do**
  $stretch[j] \leftarrow SquareParam(v_{bj}$ as $P_{0,0})$
optimum $\leftarrow$ index of $stretch[\,]$ that has best result

Algorithm 2: Pseudo-code of step-sampling

4.3.2 Experiment and Result

We did experiments on various models with various step values. Let the total test cases (boundary mappings) be m that can be calculated from the number of vertices in the first $0.25d$ interval of E_B starting from v_{b1}. This number can be considered as a calculation time of 25 percent of brute-force approach. In our experiment, we count the numbers of doing stretch-minimizing parameterization as calculation time, including both initial step sampling and deep-searching period. After all, we represent the number of calculation times as percentage of total test cases m. We also check the optimal answer to be same as brute-force or not. Table 1 shows our results.

The result shows that we could reduce percentage of the calculation times to around 17 to 60 percent of total test cases. The step value plays important roles; appropriate step value can reduce more than half of total test cases while keeping the best answer. As for a "cow" model, which was converted from originally genus 4 to genus 0 (disk topology mesh), we could not obtain appropriate disk topology mesh. That caused a strange fact that stretches are changed extremely as shown in figure 5.

Considering about parallel-computing, our step-sampling algorithm can perform well without dependency on each boundary mapping.

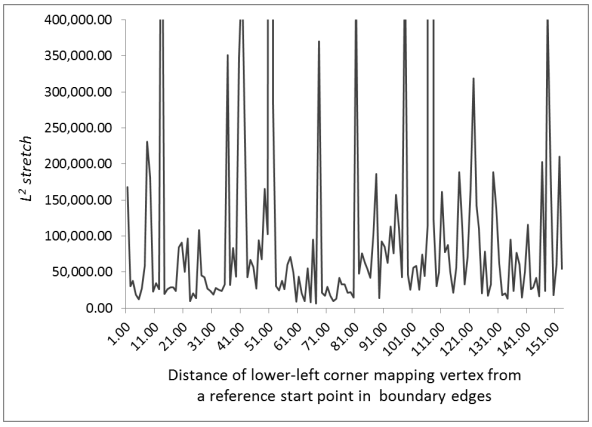


Figure 5: A graph that shows stretch values from doing square stretch-minimizing parameterization on a "cow" model that was converted from genus 4 to genus 0.

4.3.3 Step Value

From the result of step-sampling, proper step value can reduce calculation time a lot. However, too large step can result spending more calculation time than smaller one because larger step means larger searching scope.

We propose a formula for proper step value.

$$step \approx \sqrt{\frac{\text{number of vertices} \times \text{total test cases}}{\text{number of faces}}}$$

The step values from our proposed formula could reduce the calculation time to the lowest rate or nearly in most cases. Table 2 shows the results on our test models that be used same as table 1.

Model	Vertices	Faces	Total Test Cases	Step
Hand	1085	2006	18	3.12
Head	713	1357	20	3.24
Bunny Holes	36179	69463	153	8.93
Bunny No hole	35070	69644	123	7.87
Max	49342	98262	112	7.50
Cow	17135	33316	240	11.11

Table 2: Step value from our proposed formula on each testing model.

 ISBN 978-80-86943-75-6

5 CONCLUSION

The easiest way for delivering the lowest stretching square parameterization is to do brute-force approach with a stretch-minimizing parameterization method. However, it will consume a lot of calculation time.

We have presented a novel approach to optimize square parameterization that enables to reduce calculation time a lot. From our experiments, optimizing square parameterization could not be considered from faster parameterization method; instead, it should directly do stretch-minimizing method to obtain optimal result. Also, we tried to apply an existing optimization; particle swarm optimization to our problem. It still could not reduce calculation time as expectation. However, we analyzed PSO algorithm itself then applied the concept of sampling to create our approach.

We propose our "step-sampling" concept which reduces much calculation time while maintaining a stable optimal result. Although it is a simple algorithm, we can have great performance that reduce more than half from brute-force approach. We also propose a formula to calculate suitable step number based on mesh's information that will minimize the calculation time.

We are still interested to improve the performance of our "step-sampling" approach. There are many procedures that might able to be improved such as deep-searching section or changing the way of doing static stepping.

6 ACKNOWLEDGMENTS

We would like to gratefully thank all reviewers, Shin Yoshizawa for C++ code of his stretch-minimizing parameterization [Yos04a], Hugues Hoppe for filled holes bunny and hand model data.

The models are courtesy of the Stanford University (bunny), the University of Washington (head), MPI für Informatik (max) and AIM@SHAPE(cow and ustica). This work was supported by JSPS KAKENHI Grant Number 24300035.

7 REFERENCES

[Coo87a] Robert L. Cook, Loren Carpenter, and Edwin Catmull. The Reyes image rendering architecture. SIGGRAPH Comput. Graphics. 21, 4 (August 1987), 95-102.,1987.

[Des02a] M. Desbrun, M. Meyer, and P. Alliez. Intrinsic parameterizations of surface meshes. Comput. Graphics Forum, 21(3):209-218, 2002.

[Eck95a] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. Multiresolution analysis of arbitrary meshes. In Proceedings of the 22nd annual conference on Computer graphics and interactive techniques, SIGGRAPH'95, pages 173-182, 1995.

[Eri10a] M. Erik and H. Pedersen. Good parameters for particle swarm optimization. Hvass Laboratories Technical Report, HL1001, 2010.

[Flo97a] Michael S. Floater. Parametrization and smooth approximation of surface triangulations. Computer Aided Geometric Design, 14:231-250, April 1997.

[Flo03a] Michael S. Floater. Mean value coordinates. Computer Aided Geometric Design, 20:19-27, March 2003.

[Ken95a] J. Kennedy and R. Eberhart. Particle swarm optimization. In Neural Networks, 1995. Proceedings., IEEE International Conference on, volume 4, pages 1942-1948 vol.4, nov/dec 1995.

[Lev02a] B. Levy, S. Petitjean, N. Ray, and J. Maillo t. Least squares conformal maps for automatic texture atlas generation. In ACM, editor, ACM SIGGRAPH conference proceedings, Jul 2002.

[Lip12a] Lipman Yaron. Bounded distortion mapping spaces for triangular meshes. ACM Trans. Graph. 31, 4, pages 108:1–108:13, July 2012.

[San01a] P. V. Sander, J. Snyder, S. J. Gortler, and H. Hoppe. Texture mapping progressive meshes. In Proceedings of the 28th annual conference on Computer graphics and interactive techniques, SIGGRAPH'01, pages 409-416, 2001.

[Sor02a] Olga Sorkine, Daniel Cohen-Or, Rony Goldenthal, and Dani Lischinski. Bounded-distortion piecewise mesh parameterization. In Proceedings of the conference on Visualization '02. IEEE Computer Society,pages 355-362, 2002.

[Tut63a] W. T. Tutte. How to draw a graph. Proceedings of The London Mathematical Society, s3-13:743-767, 1963.

[Wik13a] Wikipedia, Mathematical optimization, http://en.wikipedia.org/wiki/Mathematical_optimization, 2013.

[Yos04a] S. Yoshizawa, A. Belyaev, and H.-P. Seidel. A fast and simple stretch-minimizing mesh parameterization. In SMI'04: Proceedings of the Shape Modeling International 2004, pages 200-208, 2004.

 ISBN 978-80-86943-75-6