Eurographics Symposium on Rendering 2011
Ravi Ramamoorthi and Erik Reinhard
(Guest Editors)

Volume 30 (2011), Number 4

Efficient Packing of Arbitrary Shaped Charts for Automatic Texture Atlas Generation

T. Nöll[1] and D. Stricker[1]

[1]German Research Center for Artificial Intelligence, Trippstadter Str. 122, D-67663 Kaiserslautern, Germany

Abstract

Texture atlases are commonly used as representations for mesh parameterizations in numerous applications including texture and normal mapping. Therefore, packing is an important post-processing step that tries to place and orient the single parameterizations in a way that the available space is used as efficiently as possible. However, since packing is NP hard, only heuristics can be used in practice to find near-optimal solutions. In this publication we introduce the new search space of modulo valid packings. The key idea thereby is to allow the texture charts to wrap around in the atlas. By utilizing this search space we propose a new algorithm that can be used in order to automatically pack texture atlases. In the evaluation section we show that our algorithm achieves solutions with a significantly higher packing efficiency when compared to the state of the art, especially for complex packing problems.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Bitmap and framebuffer operations I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

1. Introduction

Parameterizations [LPRM02, SLMB05, ZLS07] are essential when performing computations on the surface of a mesh. This includes applications in remeshing, surface reconstruction and numerous geometry processing algorithms. One prominent use of parameterizations are texture maps which provide an intuitive way of assigning information to the surface of a mesh. Information encoded in the texture are usually surface colors or lighting information such as the normals or displacement offsets.

Since three dimensional surfaces cannot be mapped to the 2D domain without any form of distortion, a natural consequence of each parameterization technique is that a certain amount of distortion is introduced. However, usually a parameterization is required that uniformly covers the surface. In practice an often used solution to bound the distortion is to segment the mesh in regions homeomorphic to discs – so called charts – that can be unfolded to the 2D domain without exceeding a certain threshold of distortion [LPRM02, SCOGL02, SWG*03, ZSGS04, JKS05]. Each chart is then parameterized separately. This imposes an additional post-processing challenge: The several parameterization domains have to be placed or *packed* together in order to share a common domain [HLS07] – the texture atlas – with a minimal amount of unused space.

It has to be mentioned that there exist also other parameterization techniques that circumvent the packing problem: In [SH02] a single chart is formed and cuts are added in order to bound the distortion. In [THCM04] the concept of cube mapping is extended to arbitrary meshes by combining multiple cube maps. In particular for complex geometry using alternative techniques it can get hard to assure a uniform distribution of the induced stretch. Thus in practice mainly texture atlases are used today, especially if artists are involved.

Being an alternative formulation of the 2D bin packing problem, chart packing is *NP* hard and thus only heuristics can be used in practice in order to efficiently find near-optimal solutions. While there exist good approximations for packing rectangles [LTWH08, MFNK95], charts usually can have arbitrary shapes in the 2D domain and thus an approximation of the chart shapes using rectangles will not provide an efficient packing (i.e. lots of space is unused).

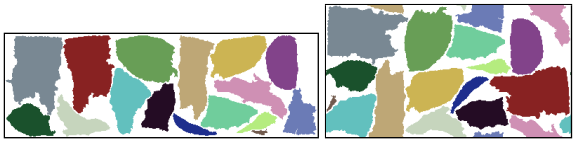
The problem of packing charts with arbitrary shapes for texturing purposes has been addressed by several authors.

To our knowledge however, there exists no publication that solely focuses on this problem until today. Lévy et al. [LPRM02] propose an approach where charts are sorted according to their size and are then dropped successively at positions that minimize a horizon line. Sander et al. [SWG*03] later generalized this approach by allowing different rotations of the charts. This enabled a more efficient packing. Zhang et al. [ZMT05] independently developed an algorithm that is very similar to Sander's. We found the most sophisticated method to be the one proposed by Zhou et al. [ZSGS04] that is similar to Sander's, but facilitates a dropping from all 4 sides of the texture domain. We implemented, evaluated and compared all methods.

All previously published methods were developed in order to pack automatically generated parameterizations. Hereby usually a large set of charts with simple convex shapes have to be packed together. State of the art methods perform well on those datasets (see Figure 1). Because of sampling artifacts that arise at the borders of the charts, parameterizations with a small number of charts are to favor. This however increases the shape complexity of the single charts. State of the art methods often fail to provide an efficient packing for such datasets (see Figures 9, 10, 11 and 12).

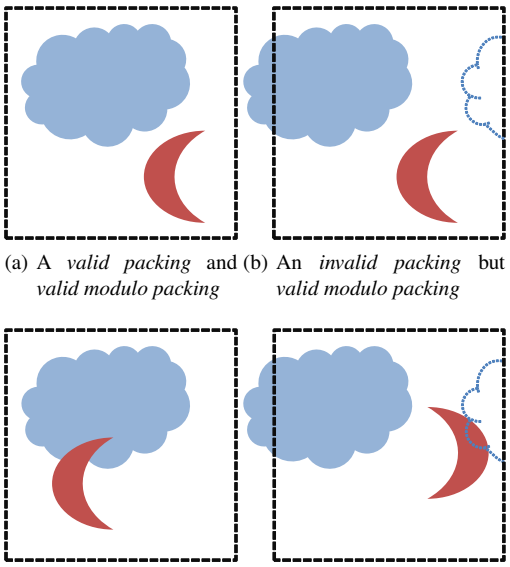
The main contribution of this paper is the introduction of a new packing algorithm for arbitrarily shaped charts. We will show in the evaluation section that our algorithm consequentially outperforms the previously published ones, especially for complex packing problems. The key idea of our algorithm is that it searches its solution in a different search space than previously published methods. In this search space the texture charts are allowed to wrap around in the atlas. Our algorithm also accounts for holes that might appear inside the parameterizations or packings.

The remainder of this paper is organized as follows: First we formally define the search space of the *valid packings*. We also introduce the new search space of the *modulo valid packings*. Then we introduce a new algorithm that finds good solutions in this search space. We perform a thorough evaluation of our algorithm and compare the results to three other methods [LPRM02, SWG*03, ZSGS04]. We finally end with a practical example and the conclusion.



(a) [SWG*03]: 67% (b) Our method: 72%

Figure 1: *Packing efficiency for the gargoyle dataset from [SWG*03]: A large number of charts with mainly convex shapes are packed to a texture atlas. State of the art methods perform well on datasets of this type.*



(a) A *valid packing* and *valid modulo packing* (b) An *invalid packing* but *valid modulo packing*

(c) An *invalid packing* and *invalid modulo packing* (d) An *invalid packing* and *invalid modulo packing*

Figure 2: *Differences between a valid packing and a valid modulo packing: A valid packing restricts each chart to stay within the texture canvas (black dashed box), while a valid modulo packing provides more flexibility. The blue dotted line represents the corresponding modulo values of the blue chart.*

2. Method

Input to our algorithm is a mesh where each vertex has been tagged with a chart number it belongs to and a 2D vector $x = (u,v)^T$ that represents the parameterization value of this vertex within the respective chart (note that vertices that lie on the boundary of two charts can be tagged multiple times). Thus, for each chart i a parameterization

$$P_i : D_i \subset \mathbb{R}^2 \mapsto S_i \subset \mathbb{R}^3$$

is induced by the parameter values of the corresponding vertices. This parameterization maps from the 2D parameter domain D_i to one part of the 3D mesh surface S_i.

Formally we define the problem of finding a *valid packing* for a given texture resolution with width w and height h as to find mappings

$$T_i : x \in \mathbb{R}^2 \mapsto s(R_i x + t_i) \in \mathbb{R}^2$$

that are applied beforehand to the parameter values x of the vertices that belong to chart i, such that the following prop-

erties are fulfilled:

$$\textit{Non overlapping:} \bigvee_{i \neq j} (D_i \cap D_j = \emptyset)$$

$$\textit{Compact packing:} \bigvee_{i} (D_i \subseteq [0 \dots w] \times [0 \dots h])$$

Where $R_i \in \mathbb{R}^{2\times 2}$ is a rotation matrix, $t_i \in \mathbb{R}^2$ is a translation vector and $s \in \mathbb{R}$ is a global scaling factor.

All previously mentioned methods search a solution in the space of *valid packings*. We instead define a different search space and we will later show why this provides better results (see Subsection 2.2).

We define a modulo function

$$M : (u,v)^T \mapsto (u \bmod w, v \bmod h)^T$$

which we extend to sets. Now we formally define the problem of finding a *modulo valid packing* as to find mappings T_i such that the following properties are fulfilled:

$$\textit{Non self-overlapping:} \bigvee_{i} (M(D_i) \text{ injective})$$

$$\textit{Non overlapping:} \bigvee_{i \neq j} (M(D_i) \cap M(D_j) = \emptyset)$$

The differences between both packing definitions are depicted in Figure 2.

Finally we define an *optimal (modulo) packing* as a *valid (modulo) packing* that maximizes the *packing efficiency*

$$\frac{\sum_i area(D_i)}{w \cdot h}$$

Summarized, our algorithm performs the following steps in order to find a good approximation for an *optimal modulo packing*:

1. A global scaling factor s is chosen (see Subsection 2.2.3).
2. The domains D_i are discretized and sorted in descending order according to their area (see Subsection 2.1).
3. The discretized domains are positioned iteratively (see Subsection 2.2.1) as long as the current packing remains a *valid modulo packing* (see Subsection 2.2.2). The values for R_i and t_i are thereby successively defined .
4. Depending on the result of step 3 we update s.
5. We iterate until a certain *packing efficiency* level τ of convergence is reached.

2.1. Discretization

Similar to [LPRM02] we also use a discretized domain since the parameterization domain will be discretized into pixels anyhow. This allows for a more efficient handling than using e.g. piecewise linear functions. [LPRM02] proposes to use the desired texture resolution for the discretization. However, this creates a direct dependence for the algorithms runtime (usually $O(n^2)$) on the desired texture resolution.

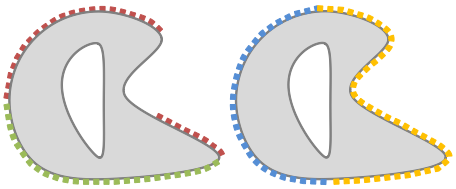


Figure 3: *A chart and its horizon lines: Top (red), bottom (green), left(blue), right (yellow) horizon lines. Note that the inner hole is not appropriately captured.*

Our observation is that the overall change in the discretized chart shapes is very small when changing the discretization resolution. Thus, the final texture resolution has only a minor effect on the *packing efficiency*. This enables to use a dual resolution approach with a significant coarser discretization. We found out empirically that a factor of 16 provides good results. Thus, if our desired texture resolution is 4096x4096 we actually work on a discretized domain with a resolution of 256x256. This provides a huge speedup without significantly reducing the *packing efficiency*.

Having discretized the parameterization domain, it is now important to find a good approximation for each arbitrarily shaped chart that enables an efficient handling. Depending on the chosen scale factor s (see Subsection 2.2.3) we raster each chart domain instance D_i using the same discretization step size than used for the overall texture resolution. Using the rasterized versions it is easy to compute for each chart domain several characteristic lines. [LPRM02, SWG*03] chose for each chart the discrete top and bottom horizon line to be the approximation of its shape. Since this can omit concave characteristics at the side of the chart, we also calculate the discrete left and right horizon line for each chart (see Figure 3). Given that for each domain we compute different rotation instances the horizon lines provide and efficient approximation of all of the charts convex and most concave characteristics. Since charts are inserted one by one it is also important to find an approximation for the current state of the packing. We therefore compute all four discrete horizon lines also for the current packing.

In Subsection 2.2.1 we present our chart placement strategy. We also show how we account for holes that might appear inside a chart and are not captured adequately using the horizon lines (see Figure 3).

2.2. Chart placement

Since packing is *NP* hard and assuming $NP \neq P$, the only way to find the *optimal packing* is to try out all combinations R_i, t_i and s that produce a *valid packing* and choose the one with the largest *packing efficiency*. This is clearly not feasible. A good heuristic thus has to be developed that finds a near optimal solution within reasonable time. Similar than [LPRM02, SWG*03, ZSGS04] we base our strategy

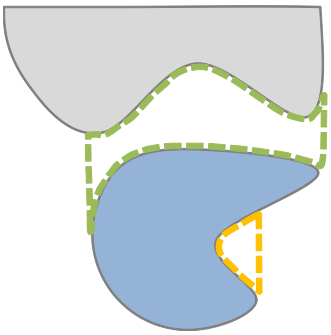


Figure 4: *The wasted space functional as defined by [SWG*03]: Depicted is the horizontal case where the blue chart is added from below to the current packing (gray). Added are the horizontal area wasted within the current packing (green) and the horizontal area wasted within the current chart (yellow) that could have been used more efficiently using another chart rotation. The vertical case is computed accordingly.*

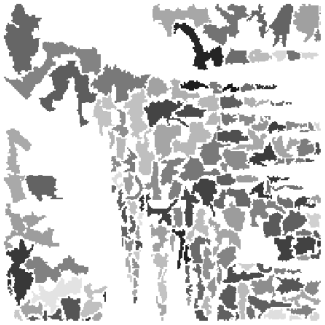


Figure 5: *Two competing placement strategies can in particular decrease the packing efficiency. Here the competition between the horizontal and vertical placement strategies create large empty bubbles inside the packing.*

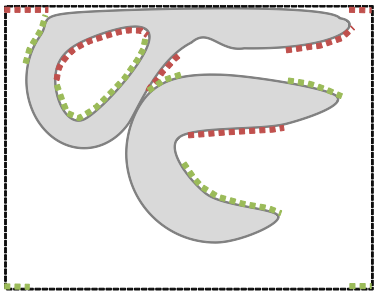


Figure 6: *Additional inner horizon lines of the current packing. Depicted is the horizon line pair for the horizontal case: Top inner horizon line (red), bottom inner horizon line (green). Additionally also the structure of holes that can exist inside a chart are identified.*

on the *best fit decreasing* strategy that has been successfully applied for the *bin packing problem*.

2.2.1. Placement strategies

The *best fit decreasing* strategy operates by first sorting the charts in decreasing order with respect to their area, and then successively inserting each chart at a position minimizing a local objective function. Like [SWG*03] we compute for each chart 16 different rotation instances. In order to find an optimal position each instance is shifted horizontally and vertically over the current packing. We do this by fixing one value of t_i and choosing the other one such that a functional is minimized. For the horizontal shifting we take the bottom horizon line as an approximation of the current packing and the charts top horizon line as an approximation for its shape. Similar for vertical shifting we use the right horizon line of the current packing and the chart's left horizon line respectively. We minimize a functional defined in [SWG*03] that measures the *wasted space* in order to find a locally optimal horizontal or vertical shifting position (see Figure 4).

The problem inherent in this preliminary strategy is that at each step either an optimal horizontal or vertical shifting position is chosen. This means that there are two competing strategies running, each of them trying to reach by iterative *local* minimizations an approximation of a *global* minimum. As depicted in Figure 5, the effect of this competition can be seen in form of large empty bubbles inside the final packing that degrade the *packing efficiency*. In order to fix this issue, we introduce two additional strategies that relax the competition between the first two ones.

Additional to the four outer horizon lines of the current packing we compute two pairs of inner horizon lines describing parts of the horizontal and vertical interior structure (see Figure 6). Each pair describes the maximal horizontal (respectively vertical) gap inside the current packing. If there exists no such interior gap we simply take the maximal gap within the current packing and the boundary of the discretization domain. Note that by utilizing these four additional horizon lines we found an efficient way to identify gaps within the current packing that would not be handled accordingly by the two first chart placement strategies on the one hand. On the other hand also holes are identified in a global manner that might exist within previously placed charts and were not appropriately captured by computing only the four outer horizon lines of the respective chart. As two additional placement strategies we now again shift the current chart horizontally and vertically through the volumes spanned by these horizon lines and try to find a valid position minimizing the *wasted space*.

Finally for all of these four strategies we include additionally the corresponding *mirrored* strategy: E.g. rather than minimizing the *wasted space* functional between the current

packing's bottom and the charts top horizon line for the horizontal case we minimize the space within the current packing's top and the charts bottom horizon line.

Summarized this yields a total number of eight concurrent running placement strategies and for each chart we insert, we take the *valid* placement that yields the smallest *wasted space* functional. In Section 2.2.2 we describe how we decide whether a placement is valid or not. Consequentially if chart i has been placed, the rasterization of the current packing is updated using $M(D_i)$.

Sequences of chart placements that take place inside the current packing are more likely to produce a compact packing. Therefore we penalize the functional values of the four strategies that place a chart on top of the current packing with a factor of λ. We used $\lambda = 1.1$ in all of our experiments.

Consequentially, for each chosen scale factor s (see Subsection 2.2.3) this placement procedure computes a unique sequence of positioning parameters R_i and t_i for each chart i until either all charts could successfully be positioned or no position could be found for the current chart that passes the placement verification test (see Subsection 2.2.2).

2.2.2. Placement verification

All texture based packing approaches that we are aware of [LPRM02, SWG*03, ZMT05, ZSGS04] try to approximate the optimal solution in the space of *valid packings*. At the beginning of the chapter we introduced the space of *modulo valid packings* that is a superspace. The advantage of using this space is that it allows for a potential much tighter packing because the restrictions that are imposed due to the boundaries of the parameterization domain are weakened. Additionally, since the corresponding texels for texture coordinates are computed modulo wise on all graphic platforms, every *modulo valid* packed texture will integrate in the rendering pipeline without introducing any sampling artifact (mipmapping, interpolation, ...) nor requiring any special treatment at the boundary of the texture domain.

Luckily, we can decide very efficiently for a current *modulo valid packing* whether a placement of a chart is *valid* depending on the placement strategy. An example is given in Figure 7: If a chart has been placed on the bottom of the current packing P using the first placement strategy, we simply check whether the bottom horizon line of this chart at the chosen position does not exceed h plus the corresponding values stored at the top horizon of P (with (w,h) being the texture dimension). Thus, if (w_i,h_i) are the discretization dimension of chart C_i under some rotation R_i, and $t_i = (x_i,y_i)^T$ is the translation chosen using the first placement strategy, then the overall originating packing is still a *modulo valid packing* iff

$$\forall_{x=0\ldots w_i} \left(bottom_{C_i|R_i}(x) + y_i < h + top_P((x+x_i) \bmod w)\right)$$

Similar rules can be derived for the other placement strategies.

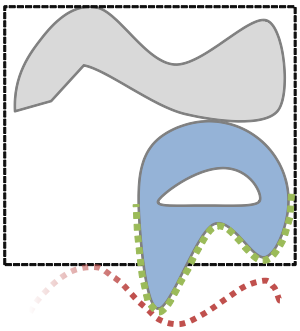


Figure 7: *Placement verification: The blue chart has been placed using the first strategy. The placement is valid since the bottom horizon line of this chart at the chosen position does not exceed the parameterization domain height plus the corresponding values stored at the top horizon of the current packing (gray).*

2.2.3. Scaling

The single parameter left to estimate is the global scaling factor s. In order to do so, we first calculate an optimal scaling factor s_{opt} assuming a *packing efficiency* of 100%:

$$s_{opt} = \sqrt{\frac{w \cdot h}{\sum_i area(D_i)}}$$

Now clearly $s = f \cdot s_{opt}$ with $f \in (0,1]$.

In order to find the missing parameter f, we perform an extended binary search over the interval $(0,1]$: We set the starting value of the lower and upper bound to $l = 0.1$ and $u = 0.9$ respectively. The current factor f is simply the median of the current upper and lower bound: $f = \frac{(l+u)}{2}$. Now we test whether a successful *modulo valid packing* could be computed using the actual scaling factor. If this is the case, we set $l = f$, otherwise we set $u = f + c$. Because the chosen scale factor also has an influence in the way charts are successively placed, we observed that in some cases a packing could be obtained for a specific scaling factor whereas it could not be obtained for a scaling factor that was slightly less. To gain robustness with respect to this observation, we therefore set $c = \frac{(u-f)}{\sigma}$. Finally we iterate until a successful *modulo valid packing* with threshold $u - l < \tau$ could be achieved. We set $\sigma = 3$ and $\tau = 0.01$ in all our experiments.

3. Evaluation

In this section we compare our method with other packing methods [LPRM02, SWG*03, ZSGS04] using different datasets. In Table 1 we summarized all evaluations.

The efficiency of each algorithm is evaluated using the *packing efficiency* as a metric. This metric describes the ratio of the used space within a texture relative to the total space

available. Intuitively, it provides information on how good the respective algorithm is working.

We also evaluated the runtime performance of each algorithm. The runtime of the placement procedure is only dependent on the discretization dimension and on the complexity of the 2D chart shapes. The placement procedures however utilize *rasterized* (i.e. discretized) versions of the chart shapes. The computation of those rasterizations is highly dependent on the mesh complexity. For better comparison we separately provide the runtime needed for rasterization only. For all evaluated algorithms we used the binary search method explained in Subsection 2.2.3 in order to determine the scale factor.

For the computation of the charts we used [SWG*03] with the following modification: In order to generate complex packing problems we iteratively split and merge charts until the *stretch* induced by each parameterization does not exceed a threshold of 2.0. It can be seen in the example datasets that this procedure produces a small number of charts with complex shapes. Finding an efficient packing for those datasets is challenging. For the parameterization itself we used the method proposed in [ZLS07]. In all our experiments we used a discretization with a resolution of 256^2. Note that this coarse resolution is sufficient in order to effectively compute the corresponding position parameters s, R_i and t_i for each chart i. Using the computed placement parameters from the coarse discretization, it is easy to assemble a high resolution discretization domain that has almost the same *packing efficiency*.

We test for 31 texture aspect ratios ranging from $91x724 \ldots 256x256 \ldots 724x91$ and keep the packing where the best result could be achieved. Additionally we provide results where we forced the texture domain to be square.

In order to provide a significant evaluation we utilized a mesh benchmark consisting of 380 meshes from 19 categories [CGF09].

3.1. Analysis

In Figures 9, 10, 11 and 12 we exemplarily show packing results that the different algorithms provided. It can be seen that our algorithm consequentially uses the available space more efficiently and thus produces results with a higher *packing efficiency* than the compared methods [LPRM02,SWG*03,ZSGS04]. Table 1 summarizes the evaluation of all 380 packing sequences. Remarkable is that in 98.7% of the experiments our method provided the best packing efficiency of all evaluated algorithms. As can be seen in Figure 8 our algorithm is up to a certain degree independent of the restrictions imposed by the texture aspect ratio.

The packing (i.e. the determination of the placement parameters) itself took on our machine (Intel® Xeon W3550 @ 3.07GHz) for all evaluated algorithms only several seconds. Most of the time was spent rasterizing the mesh charts at different rotations and scales. Note that we use a very simple and inefficient barycentric coordinates approach in order to check whether a pixel lies within a triangle or not. The time spent here could easily be reduced using more advanced triangle rasterization methods which is however not the focus of our publication.

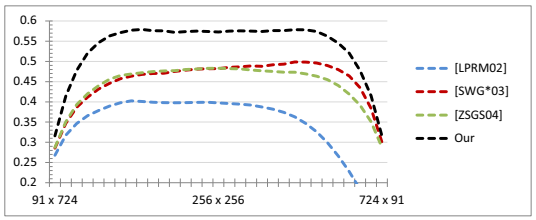


Figure 8: *Average packing efficiency (left) depending on the texture aspect ratio (bottom): Our algorithm consequentially uses the available space more efficiently. By searching for a solution in the modulo packing space, our algorithm is up to a certain degree independent of the texture aspect ratio.*

3.2. Application

In Figure 13 we conclude the evaluation with a practical application of texture packing: For the *armadillo* dataset we compute a shape preserving simplified version of the mesh [LT98] where the number of faces has been reduced to only 1% of the original face count. The lost information is stored in form of a 256x256 normal map [SGG*00]. The limited amount of space in the normal map requires a very efficient packing. Packing this information using [LPRM02] gives significantly worse results than using our method.

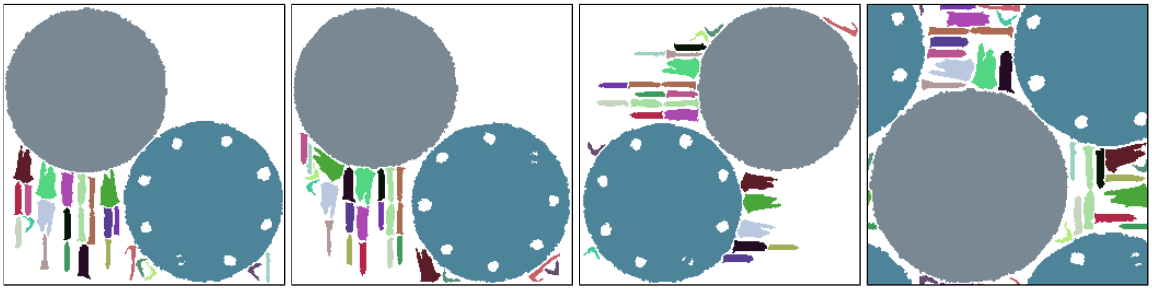
4. Conclusion

In this publication we introduced a new search space of packings, the space of *modulo valid packings*. We showed that while enabling more efficient packings, solutions in this space seamlessly integrate into the texture rendering pipeline without the need for adjustments or special treatments. We developed a new efficient algorithm that finds good approximations of *optimal modulo packings*. By extensive evaluation we showed that our algorithm provides a significant higher *packing efficiency* when compared to other published methods.

Yet we think that there is room for improvement: We therefore plan to not only use the 2D information of the rasterized charts, but also to introduce properties of the corresponding mesh. Neighboring charts on the mesh often have parts in the rasterization where they share very similar border characteristics. Those characteristics might not be appropriately treated by our placement strategies even though

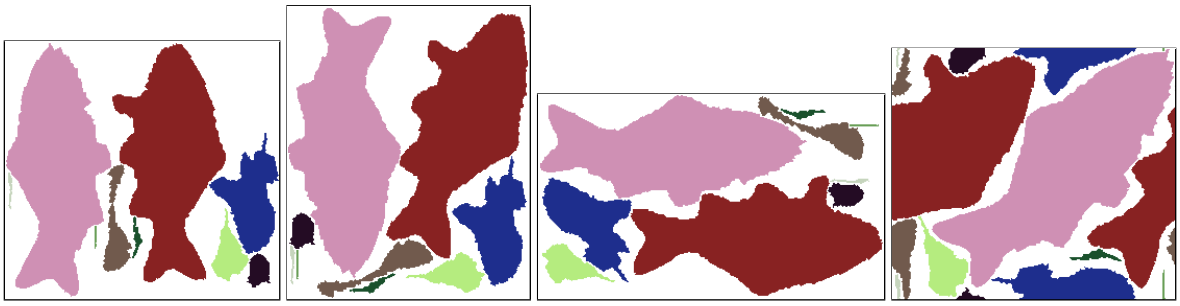
Method:	[LPRM02]	[SWG*03]	[ZSGS04]	Our
Variable aspect ratio				
Achieved best efficiency (%):	0	0.8	0.5	98.7
Avg. efficiency (%):	45	54	53	63
Min. efficiency (%):	0.17	0.21	0.19	0.26
Max. efficiency (%):	0.90	0.91	0.91	0.92
Avg. rasterization time (s):	1.91	29.03	26.99	36.16
Avg. total time (s):	5.45	38.11	34.25	55.35
Square aspect ratio				
Achieved best efficiency (%):	0.3	4.5	1.3	93.9
Avg. efficiency (%):	40	48	48	57
Min. efficiency (%):	6	13	13	13
Max. efficiency (%):	78	77	78	84
Avg. rasterization time (s):	0.19	3.27	3.06	4.08
Avg. total time (s):	0.29	3.58	3.44	4.70

Table 1: *Evaluation results from the from the Watertight Track benchmark developed at IMATI-CNR [GBP07] consisting of 380 meshes (see Chapter 3 for further explanation).*



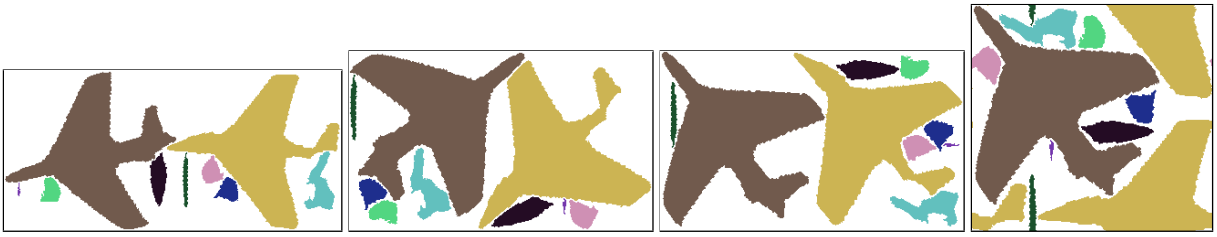
(a) [LPRM02]: 58% (b) [SWG*03]: 58% (c) [ZSGS04]: 58% (d) Our method: 84%

Figure 9: *Packing efficiency of the table dataset. The texture aspect ratio is forced to be square in this example.*



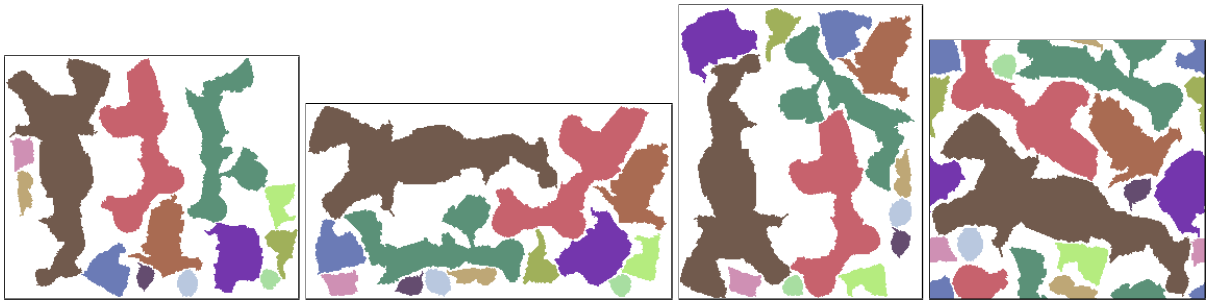
(a) [LPRM02]: 53% (b) [SWG*03]: 61% (c) [ZSGS04]: 60% (d) Our method: 77%

Figure 10: *Packing efficiency of the fish dataset*



(a) [LPRM02]: 39% (b) [SWG*03]: 53% (c) [ZSGS04]: 53% (d) Our method: 69%

Figure 11: *Packing efficiency of the airplane dataset*



(a) [LPRM02]: 45% (b) [SWG*03]: 54% (c) [ZSGS04]: 52% (d) Our method: 68%

Figure 12: *Packing efficiency of the shapes dataset*

a placement with a *wasted space* of almost zero could be achieved here. There might also be better solutions to find the optimal texture aspect ratio than performing a linear search. We think that the runtime performance and the *packing efficiency* of our method could be further improved if the texture aspect ratio would be adjusted on-the-fly during the packing procedure.

5. Acknowledgements

The benchmark we used for evaluation is based on a set of polygonal models from the *Watertight Track* of the *2007 SHREC Shape-based Retrieval Contest* (http://www.aimatshape.net/event/SHREC/shrec07), developed at IMATI-CNR [GBP07]. The *armadillo* dataset is provided generously by the *Stanford University Computer Graphics Laboratory* (http://graphics.stanford.edu/data/3Dscanrep). This work has been partially funded by the project CAPTURE (01IW09001).

References

[CGF09] CHEN X., GOLOVINSKIY A., FUNKHOUSER T.: A benchmark for 3d mesh segmentation. In *ACM SIGGRAPH 2009 papers* (New York, NY, USA, 2009), SIGGRAPH '09, ACM, pp. 73:1–73:12. 6

[GBP07] GIORGI D., BIASOTTI S., PARABOSCHI L.: *Watertight Models Track, technical Report IMATI-CNR-GE 09/07*. Tech. rep., 2007. 7, 8

[HLS07] HORMANN K., LÉVY B., SHEFFER A.: Mesh parameterization: theory and practice. In *ACM SIGGRAPH 2007 courses* (New York, NY, USA, 2007), SIGGRAPH '07, ACM. 1

[JKS05] JULIUS D., KRAEVOY V., SHEFFER A.: D-charts: Quasi-developable mesh segmentation. In *Computer Graphics Forum, Proceedings of Eurographics 2005* (Dublin, Ireland, 2005), vol. 24, Eurographics, Blackwell, pp. 581–590. 1

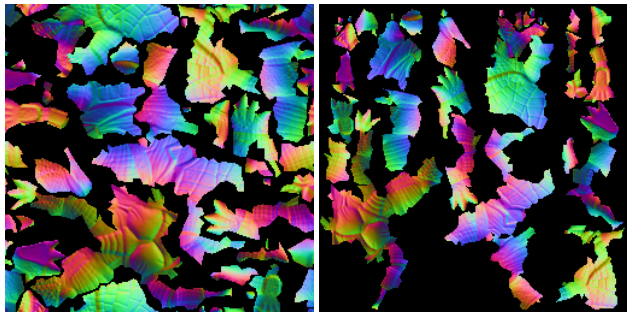
[LPRM02] LÉVY B., PETITJEAN S., RAY N., MAILLOT J.: Least squares conformal maps for automatic texture atlas generation. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2002), SIGGRAPH '02, ACM, pp. 362–371. 1, 2, 3, 5, 6, 7, 8, 9

[LT98] LINDSTROM P., TURK G.: Fast and memory efficient polygonal simplification. In *Proceedings of the conference on Visualization '98* (Los Alamitos, CA, USA, 1998), VIS '98, IEEE Computer Society Press, pp. 279–286. 6

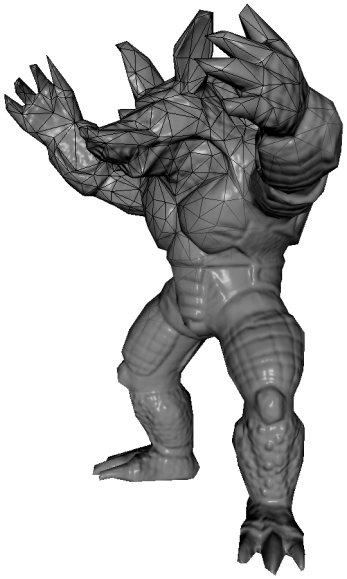
[LTWH08] LI G.-S., TRICOCHE X., WEISKOPF D., HANSEN C. D.: Flow charts: Visualization of vector fields on arbitrary surfaces. *IEEE Transactions on Visualization and Computer Graphics 14* (September 2008), 1067–1080. 1

[MFNK95] MURATA H., FUJIYOSHI K., NAKATAKE S., KAJITANI Y.: Rectangle-packing-based module placement. In *Proceedings of the 1995 IEEE/ACM international conference on Computer-aided design* (Washington, DC, USA, 1995), ICCAD '95, IEEE Computer Society, pp. 472–479. 1

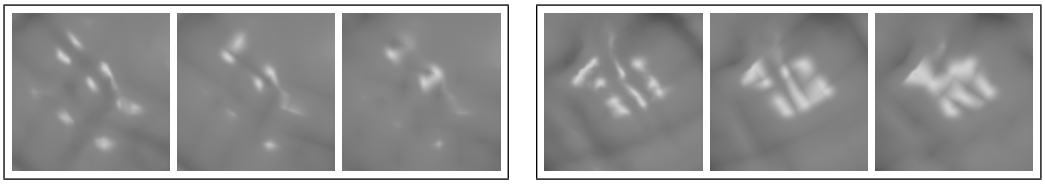
[SCOGL02] SORKINE O., COHEN-OR D., GOLDENTHAL R., LISCHINSKI D.: Bounded-distortion piecewise mesh parameterization. In *Proceedings of the conference on Visualization '02*



(a) Packed 256x256 normal maps: Our method (left), Lévy et al. [LPRM02] (right): The available space is used more efficiently by our method.



(b) Rendering of the simplified mesh using normal maps: No seams or artifacts are visible due to the usage of a *modulo packed* normal map.



(c) Close comparison of two features in the rendering (left: Ground truth, middle: normal map packed using our method, right: normal map packed using the method proposed by Lévy et al. [LPRM02]). As can be seen, our packing scheme uses the very limited space in the normal map more efficiently. Consequentially the features are closer to the ground truth when rendered.

Figure 13: *A practical application where our method was used for packing a normal map*

(Washington, DC, USA, 2002), VIS '02, IEEE Computer Society, pp. 355–362. 1

[SGG*00] SANDER P. V., GU X., GORTLER S. J., HOPPE H., SNYDER J.: Silhouette clipping. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2000), SIGGRAPH '00, ACM Press/Addison-Wesley Publishing Co., pp. 327–334. 6

[SH02] SHEFFER A., HART J. C.: Seamster: inconspicuous low-distortion texture seam layout. In *Proceedings of the conference on Visualization '02* (Washington, DC, USA, 2002), VIS '02, IEEE Computer Society, pp. 291–298. 1

[SLMB05] SHEFFER A., LÉVY B., MOGILNITSKY M., BOGOMYAKOV A.: Abf++: fast and robust angle based flattening. *ACM Trans. Graph. 24* (April 2005), 311–330. 1

[SWG*03] SANDER P. V., WOOD Z. J., GORTLER S. J., SNYDER J., HOPPE H.: Multi-chart geometry images. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing* (Aire-la-Ville, Switzerland, Switzerland, 2003), SGP '03, Eurographics Association, pp. 146–155. 1, 2, 3, 4, 5, 6, 7, 8

[THCM04] TARINI M., HORMANN K., CIGNONI P., MONTANI C.: Polycube-maps. In *ACM SIGGRAPH 2004 Papers* (New York, NY, USA, 2004), SIGGRAPH '04, ACM, pp. 853–860. 1

[ZLS07] ZAYER R., LÉVY B., SEIDEL H.-P.: Linear angle based parameterization. In *Proceedings of the fifth Eurographics symposium on Geometry processing* (Aire-la-Ville, Switzerland, Switzerland, 2007), Eurographics Association, pp. 135–141. 1, 6

[ZMT05] ZHANG E., MISCHAIKOW K., TURK G.: Feature-based surface parameterization and texture mapping. *ACM Trans. Graph. 24* (January 2005), 1–27. 2, 5

[ZSGS04] ZHOU K., SYNDER J., GUO B., SHUM H.-Y.: Iso-charts: stretch-driven mesh parameterization using spectral analysis. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing* (New York, NY, USA, 2004), SGP '04, ACM, pp. 45–54. 1, 2, 3, 5, 6, 7, 8