DOI: 10.1111/cgf.12428
Eurographics Symposium on Geometry Processing 2014
Thomas Funkhouser and Shi-Min Hu
(Guest Editors)

Volume 33 (2014), Number 5

Efficient encoding of texture coordinates guided by mesh geometry

Libor Váša and Guido Brunnett

Chemnitz University of Technology, Chemnitz, Germany

Abstract

In this paper, we investigate the possibilities of efficient encoding of UV coordinates associated with vertices of a triangle mesh. Since most parametrization schemes attempt to achieve at least some level of conformality, we exploit the similarity of the shapes of triangles in the mesh and in the parametrization. We propose two approaches building on this idea: first, applying a recently proposed generalization of the parallelogram predictor, using the inner angles of mesh triangles corresponding to the UV-space triangles. Second, we propose an encoding method based on discrete Laplace operator, which also allows exploiting the information contained in the mesh geometry to efficiently encode the parametrization. Our experiments show that the proposed approach leads to savings of up to 3 bits per UV vertex, without loss of precision.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Line and curve generation

1. Introduction

The amount of details captured in 3D models rises constantly in the recent years, and this trend is expected in the future as well. The high resolution of the meshes used to represent the models leads directly to high storage and transmission costs. In order to alleviate these costs, a variety of compression and simplification algorithms has been proposed in the last two decades.

Rendering of meshes usually requires additional data describing the surface, such as surface color, reflectivity and other photometric values. These values can be associated with vertices, but that in turn leads to even higher demands on the mesh resolution. A common approach to alleviating this problem is to compute a mapping of the surface into a 2D space, and express the surface properties as a 2D texture, i.e. an sampled image. This in turn allows using a lower number of vertices. Recently, approaches such as normal mapping or displacement mapping are being used to capture even geometric details of the surface in the form of a texture. This means that in practice it is possible to display a detailed 3D object represented with a relatively small number of vertices, yet at the cost of needing additional data associated with each vertex. Compression of this additional data is the topic of our contribution.

A textured mesh is described by its connectivity (mesh triangles), geometry (3D positions of mesh vertices), a texture image and a mapping that describes the parameterization of the 3D surface in the 2D space. This mapping is usually expressed in terms of a set of 2D vertices (we call them *texture vertices*, or *texture geometry* in general), and a correspondence map, that assigns a texture vertex to each corner of each mesh triangle. These mapped triangles will be referred to as *texture triangles*. In general, there may be up to 6 times as many texture vertices as there are mesh vertices, which corresponds to the situation where a unique texture vertex is associated to each triangle corner in the mesh. In practice, however, only a single texture vertex is usually used for all corners incident to a particular mesh vertex, making the number of mesh vertices and texture vertices roughly equal. Having two coordinates per texture vertex makes up for a considerable amount of data that have to be stored. In this paper, we address the issue of efficient encoding of the texture vertices, and we also touch on the encoding of the mapping between mesh triangles and texture triangles.

There are many ways of computing a parameterization of a given triangle mesh, with different properties. Ideally, the parameterization would preserve angles (conformality) and areas, but such so-called isometric parameterization can be only found for a very small class of unfoldable meshes. Generally, parameterization algorithms attempt to find a balance between preservation of angles and areas, and artists who create parameterizations manually also follow these goals intuitively.

Regardless of the particular method used to compute the parameterization, it is therefore reasonable to expect a certain level of similarity between the mesh triangles and their corresponding texture triangles in practice. This similarity manifests itself in turn as certain redundancy in texture data (texture vertex positions) with respect to mesh data (mesh vertex positions). Our aim is to exploit this similarity, reducing the redundancy by prediction, thus reducing the amount of data required to describe the parameterization.

In a practical scenario, the encoding of a textured mesh is done in following steps:

- the mesh connectivity is encoded,
- the coordinates of mesh vertices are encoded,
- the texture connectivity is encoded,
- the coordinates of texture vertices are encoded.

Hence we assume, that at the point when the texture connectivity and geometry is being decoded, the mesh geometry and connectivity is already available at the decoder side. This allows us to use it in our algorithms.

In this paper, we present two approaches building on the idea of exploiting the similarity between mesh geometry and texture geometry:

1. We use the recently proposed generalization of the parallelogram predictor to predict the coordinates of the texture vertices. The form of the predictor allows us to use the knowledge of the shapes of the mesh triangles to improve the accuracy of the prediction and thus reduce the data rate. This procedure will be described in section 4.
2. We employ the discrete Laplace operator to transform the texture coordinates. Knowledge of the mesh vertex positions allows us to build a Laplace operator that produces residuals of lower entropy, thus reducing the required data rate. This procedure will be described in section 5.

The first approach is easy to implement and fast to execute, while the second one provides lower data rates in some situations, at the cost of introducing the necessity of solving a large sparse system of linear equations at the decoder. The comparison of the two approaches and the state of the art will be given in section 6.

Apart form these two algorithms that encode the coordinates of texture vertices, we also revisit the problem of encoding of texture connectivity, describing a simplified approach that can be used in most practical scenarios and that turns out to be more efficient than some of the state of the art approaches. The encoding scheme is described in section 3.

2. Related Work

The problem of encoding of texture connectivity and geometry did attract some attention of researchers in the past, although not as much as the related problem of mesh encoding. These two problems are commonly addressed in a joint fashion, together with other per vertex or per corner properties, such as vertex normals or vertex colors. Our proposed approach builds on methods suggested for mesh encoding, however, it is independent of the particular method used for encoding of the mesh. Our proposal focuses on how to modify the mesh encoding techniques in order to exploit the mesh connectivity and geometry data that is available at the decoder.

There exists a variety of methods for encoding of mesh connectivity. Starting from the encoding of vertex spanning tree and triangle spanning tree in the Topological Surgery algorithm by Taubin and Rossignac [TR98], through the Edgebreaker algorithm with guaranteed upper bound of 2 bits per triangle by Rossignac [Ros99] up to to the valence driven algorithms by Touma and Gotsman [TG98] and Kälberer et al. [KPRW05], the algorithms are getting quite close to the theoretical limit of 3.245 bits per triangle, sometimes even improving on it by exploiting the uneven probability distribution of possible connectivities. In the current state of the art, the encoded connectivity usually represents about 5-15% of the data rate required for encoding a triangle mesh, with the rest being occupied by the encoded vertex coordinates.

In order to encode the mesh vertex positions efficiently, it is common to apply the predict-correct approach. In this scenario, the decoder forms a prediction of the value that is about to be decoded from the data stream, based on the data available to it (i.e. from the positions of previously decoded vertices). The encoder mimics the prediction and instead of the value itself, it only encodes the difference between the prediction and the value, called correction or residual. The decoder then adds the correction to the prediction, obtaining the desired value. If the predictions are reasonably accurate, then there appears a peak in the probability distribution of the residuals around zero. The higher is the accuracy of the prediction, the steeper is the peak, and the lower is the entropy of the residuals. The reduction of entropy is then finally exploited by an entropy coder, such as the Huffman coder [Huf52] or an adaptive arithmetic coder [MWS03]. Most of the mesh compression algorithms build on this general approach in one way or another, we are only going to mention three approaches, because they represent the building stones for our encoding scheme for texture coordinates.

The most commonly used predictor of this kind is the parallelogram predictor, introduced by Touma and Gotsman [TG98]. The prediction v'_O of a vertex v_O is based on positions of three vertices v_L, v_R and v_B, which build an adjacent triangle that has been previously decoded. The prediction has a simple form $v'_O = v_L + v_R - v_B$, known as the parallelogram rule, as it can be geometrically interpreted as building a parallelogram from the known triangle (see fig. 4). Note that values with a prime denote here and in the following a prediction or estimation of the corresponding value.

Recently, this approach has been generalised by Váša and Brunnett [VB13] to incorporate an estimation of the inner angles in the prediction stencil. The algorithm first estimates the inner angles in the prediction stencil from the mesh connectivity, and then uses a weighted extension of the parallelogram rule. This leads to an improved accuracy of the prediction and in turn to a reduction of the required data rate.

Another approach to mesh encoding has been taken by Sorkine et al [SCOT03]. Their high-pass encoding can be interpreted in terms of the predict-correct scenario as predicting each vertex in the barycentre of its 1-ring neighbourhood. The corrections with respect to such predictions are being quantized and encoded, however, in order to reconstruct the original vertex positions, it is necessary to solve a sparse system of linear equations which inverts the process of residual computation.

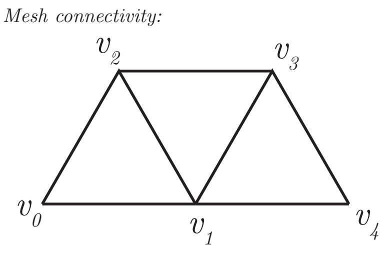
Encoding of texture connectivity can be achieved by applying any of the mesh connectivity encoders, given that the respective conditions are met. In particular, the connectivity coders usually only handle manifold connectivities, i.e. connectivities where the one ring neighbourhood of each vertex is formed by a closed disk of triangles or by a single triangle fan.

Researchers have also noted that in most practical cases, the texture connectivity is closely related to the mesh connectivity. There is always a one-on-one correspondence between the mesh triangles and texture triangles, and also the texture connectivity can in most cases be derived from the mesh connectivity by cutting some of the edges. An edge in the mesh connectivity, that is split into two edges in the texture connectivity is denoted as crease edge (see fig. 1).

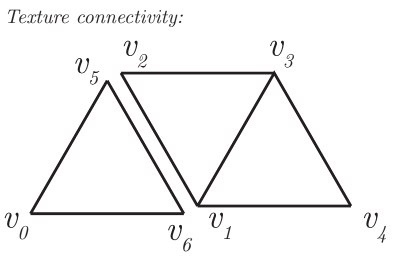
Gumhold and Straßer [GS98] suggest encoding one bit per inner edge in the mesh connectivity, which determines, whether this edge is a crease edge or not. For crease edges, two additional bits get encoded, which for each endpoint of the edge determine whether the endpoint is represented by just a single texture vertex or by two texture vertices, hence at least one of these two bits is always true.

A slightly different approach has been taken by Taubin [THLR98], who suggests encoding one bit per vertex, which determines, whether there is any crease edge incident with it. In case the bit indicates presence of crease edges, a series of additional bits is emitted, indicating for each edge emanating from the vertex whether it is a crease edge or not.

This approach has been then improved upon by Isenburg and Snoeyink [IS00], who notice that there are some cases when the bits themselves can be predicted. For example, if a vertex of degree n is determined to have crease edges emanating from it, and the first $n-1$ edges are determined not to be the crease edges, then the last bit does not have to be emitted, since it certainly must indicate that the n-th edge is a crease edge.



0	1	2
2	1	3
1	4	3



0	6	5
2	1	3
1	4	3

Figure 1: *Crease edge. The rows of the tables represent triangles in the different versions of the connectivity.*

Most of the approaches mentioned so far employ the parallelogram rule for prediction of coordinates of texture vertices. It has been noted by Isenburg and Snoeyink [IS03], that this kind of prediction is naturally highly inaccurate for the crease edges, and they have suggested using a different predictor for such cases, together with a different context of an adaptive arithmetic encoder, so that the context of the encoder does not get disrupted by the sudden change in prediction accuracy.

Finally, since the objective is to provide the decoder with a mesh parameterization that is compatible with a particular texture, it is also possible to achieve that by letting the decoder compute the parameterization directly from the mesh, while the encoder only warps the input texture so that it matches the mapping generated by the decoder [SCO01]. Such scenario does not require any data to describe the parameterization, on the other hand it suffers from problems such as quality loss of the texture due to the warping and performance issues following from the need to construct the mapping at the decoder.

3. Encoding of UV connectivity

In order to transmit the texture geometry, we need the texture connectivity to be available at the decoder side. Our approach is independent of the algorithm used for texture connectivity transmission, i.e. any of the algorithms mentioned in the previous section can be used. In our implementation, we use a simplified strategy that allows for efficient coding, given that the texture connectivity is manifold.

Our approach, similarly to previous proposals, builds on the idea that the texture connectivity can be derived from the mesh connectivity by identifying the set of crease edges. Indeed, this is the case for most practical parameterization approaches, which usually constitute of a mesh cutting algorithm, such as the Seamster [SH02], which cuts some of the edges, followed by a particular unfolding procedure [HLS07], which does not alter the connectivity. In our approach we thus encode a single bit per mesh edge, identifying whether or not the edge is a crease edge. In contrast with the approach by Gumhold and Straßer, we do not encode additional bits in the case of crease edges. This approach is motivated by the fact that only one of following cases can occur for a crease edge:

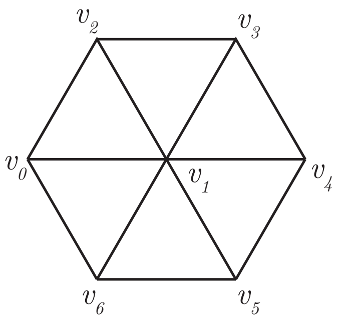
1. a vertex v is incident with a single crease edge. In that case, the additional bits would indicate "single vertex" at the endpoint v, and "two vertices" on the other endpoint (if there was a "single vertex" on the other endpoint, then the edge would not be a crease edge). Storing the flags is therefore not necessary (see Figure 2A).
2. a vertex v is incident with more then one crease edge. In that case, each of the crease edges would be flagged "two vertices" at the endpoint v. Therefore, the flags again do not have to be stored explicitly. Cases when some of the endpoints were marked as "single vertex" and others as "two vertices" always lead to a complex vertex in the texture connectivity (see Figure 2B). In such case, we resort to duplicating the vertex, because we require manifold texture connectivity for the following step.

In our approach, the texture connectivity is always manifold, which allows traversing it instead of the mesh connectivity, contrasting with the previous approaches. This allows us to avoid the problems of choosing a proper predictor for the crease edges, since the traversal never performs a prediction across a crease edge. Our proposed single edge bit encoding represents some limitation on the possible texture connectivities that can be expressed, this is however true for the other texture connectivity encoding algorithms as well. In particular, none of the algorithms allows having two texture triangles adjacent, if they are not adjacent in the mesh connectivity. This problem is equally rare in practice.

It has been previously argued [IS00] that it is more efficient to encode a single crease bit per vertex, which indicates whether or not there are any crease edges incident with that edge, and then only transmit the crease bits for the incident edges if it is indicated that there are any. Unfortunately, the comparison with the single edge bit has been done using a different encoding scheme for the bits - for the edge bits, direct encoding (1 bit per flag) has been used, while the vertex bits have been encoded using an adaptive arithmetic coder, which allowed exploiting their probability distribution. We argue, that if arithmetic coding is used for the edge bits, then it theoretically outperforms the vertex bit strategy.

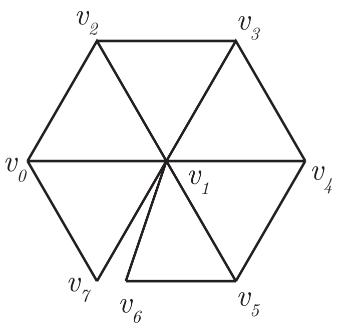
Denote p_c the ratio of crease edges in a mesh. Encoding

Mesh connectivity:



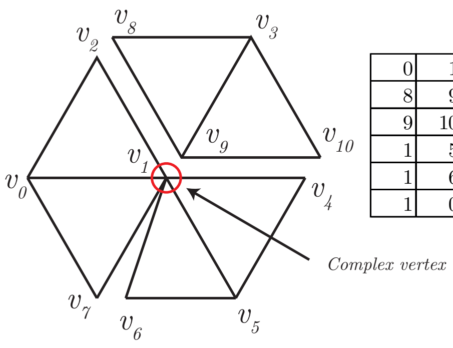
0	1	2
2	1	3
1	4	3
1	5	4
1	6	5
1	0	6

A: Texture connectivity (crease edge <1-6>):*



0	1	2
2	1	3
1	4	3
1	5	4
1	6	5
1	0	7

B: Texture connectivity (crease edges <2-1>, <1-4>, <1-6>):*



0	1	2
8	9	3
9	10	3
1	5	4
1	6	5
1	0	7

Figure 2: *Crease edges. Asterisk denotes crease edge endpoints marked as "single vertex" by the Gumhold and Straßer algorithm [GS98]. The rows of the tables represent triangles in the different versions of the connectivity.*

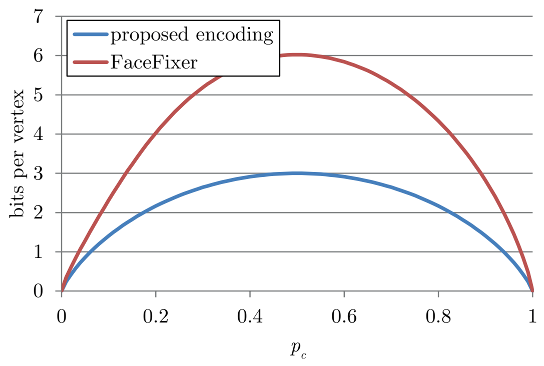


Figure 3: *Theoretical data rates for texture connectivity.*

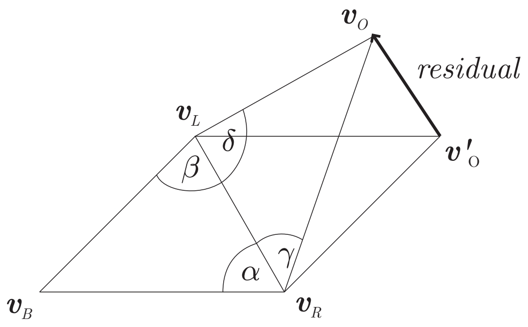


Figure 4: *Prediction stencil.*

the edge crease bits amounts to encoding values of entropy

$$H_{ce} = -\Big(p_c log_2(p_c) + (1-p_c) log_2(1-p_c)\Big). \quad (1)$$

In total, roughly $3V$ such flags are required to encode the texture connectivity (V is the number of vertices, in connected triangle meshes there are roughly $3V$ edges). Therefore, the expected bitrate per vertex is $b_{ce} = 3H_{ce}$.

In contrast to that, the probability of a vertex being a crease vertex can be estimated as $p_{cv} = 1-(1-p_c)^6$, since we can assume six edges incident with a vertex on average. The entropy of such vertex flag is

$$H_{cv} = -\Big(p_{cv} log_2(p_{cv}) + (1-p_{cv}) log_2(1-p_{cv})\Big). \quad (2)$$

Additionally, for each crease vertex, roughly 6 crease edge bits have to be encoded. Therefore, the total number of bits per vertex needed for this approach is $b_{cv} = H_{cv} + 6p_{cv}H_{ce}$. Figure 3 compares the bitrates for varying probability of crease edges. It can be seen, that the crease edge bit scenario outperforms the crease vertex bit scenario for any probability p_c.

After the connectivity is decoded, it can be traversed in a fashion similar to the Edgebreaker algorithm, i.e. adding one triangle to the processed part of the mesh at the time. If the texture coordinates of the tip vertex are not yet known to the decoder, then they are predicted and encoded/decoded. The next section deals with the prediction.

4. Encoding of texture geometry

A natural choice for the predictor of the texture vertex coordinates is the parallelogram predictor. It is possible to use the same formulation of the predictor as in the case of mesh vertex prediction, only this time working with 2D UV coordinates instead of 3D XYZ coordinates. Additionally, one can expect that the residuals will be smaller in the UV case, because there is one degree of freedom less: in the 2D case, the prediction stencil is always planar, and therefore the residuals do not have any normal component.

The key observation of our algorithm is that at this point, we can assume that the mesh geometry has been already decoded, and thus it can be used to improve the prediction of the UV coordinates. Most mesh unfolding algorithms strive to preserve the shapes of mesh triangles in the texture space. Therefore the shapes of the mesh triangles can be used to guide the predictor of the UV vertex coordinates.

Our proposal is to use the generalized weighted predictor, proposed recently by Váša and Brunnett [VB13]. It uses for each prediction two additional weights w_1 and w_2 that influence the desired shape of the prediction stencil, hence it is no longer necessarily a parallelogram. The weighted predictor takes following form:

$$v'_O = w_1 v_L + w_2 v_R + (1 - w_1 - w_2) v_B. \quad (3)$$

The weights are determined from an estimation of the inner angles in the prediction stencil. For a prediction stencil depicted in figure 4, the weights are:

$$\begin{aligned} w_1 &= \frac{\cot(\beta')+\cot(\delta')}{\cot(\delta')+\cot(\gamma')}, \\ w_2 &= \frac{\cot(\alpha')+\cot(\gamma')}{\cot(\delta')+\cot(\gamma')}. \end{aligned} \quad (4)$$

For a derivation of the weights see [VB13].

While for mesh encoding, the angles α, β, γ and δ are estimated from the vertex degrees, we propose a different approach. Since there is a bijective correspondence between the mesh triangles and texture triangles, it is possible to determine the corresponding pair of mesh triangles for each prediction stencil used in texture coordinates encoding. The inner angles of these mesh triangles are known to the decoder, and they can be used as estimates of the inner angles in the texture space prediction stencil.

The process of using angles from mesh triangles as estimates of inner angles of texture triangles builds on the assumed (at least partial) conformality of the parameterization. Conformality, however, does not imply equivalence of the inner angles, but merely preservation of angles in the tangential plane of the surface. Therefore, it makes sense to project the angles from the surface onto the tangential plane, and use the projection as estimate of inner angles in texture triangles. The projection can be done in many sophisticated manners, in our experiments we however used a simple normalization approach. For each vertex v_i, we compute the sum of its incident inner angles

$$a_i = \sum_{j \in N_c(i)} \phi_j, \tag{5}$$

where $N_c(i)$ is the set of inner corners incident with v_i, and ϕ_j is he inner angle in the given corner. Finally, for the computation of weights, we use angles $\theta_j = (2\pi\phi_j)/a_i, j \in N_c(i)$ for inner vertices. Inner angles incident with border vertices of the texture connectivity are not normalised, i.e. $\theta_j = \phi_j$.

The results of using the weighted predictor will be discussed in section 6. There is, however, a different way of encoding the texture geometry, based on discrete Laplace operator. This will be discussed next.

5. Laplacian encoding of UV coordinates

In the previous section, we have discussed prediction of texture coordinates one by one during a traversal of the texture connectivity. In contrast to that, in this section, a method will be discussed, which can be interpreted as predicting all the texture coordinates at once, using one ring neighbourhood. This approach is based on the high-pass encoding of triangle meshes, proposed by Sorkine at al. [SCOT03]. The original idea can be interpreted as predicting each vertex (mesh or texture) to lie at the barycentre of its 1-ring neighbourhood:

$$v_i' = \frac{1}{\|N(i)\|} \sum_{j \in N(i)} v_j, \tag{6}$$

where $N(i)$ is the set of all 1-ring neighbours of v_i.

It is not difficult to compute the residuals for such prediction, however, reconstructing the original coordinates from the residuals is more difficult, because it cannot be done in the one-by-one fashion. One possibility to do that is to stack all the linear equations used to construct the residuals $r_i = v_i' - v_i$ into a matrix L. This matrix is in fact equivalent to the combinatorial Tutte Laplacian of the mesh. Multiplying the vector of texture vertex coordinates by this matrix produces a vector of so-called delta coordinates, which are in fact equivalent to the residuals.

On its own, the matrix L it is not invertible, unless additional rows representing the so-called anchor vertices are added to the linear system. Each such row contains a single "1" at a column corresponding to the anchor vertex, making the coordinates of the anchor vertex part of the vector of

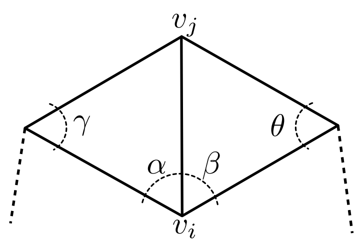


Figure 5: *Local mesh geometry.*

residuals and enforcing them in the reconstruction process. This way a rectangular matrix L^* is produced, which is solvable it in the least squares sense, obtaining the coordinates from the residuals. The strategy for the choice of anchor points is to select a random one for each connected component, and then perform an intermediate reconstruction at the encoder, identifying the vertex with the largest reconstruction error. This vertex is then used as an additional anchor vertex and the process is repeated until the required number of anchor points is reached. For more information on such approach to mesh encoding and details on anchor selection, see [SCOT03] and [CCOST05].

Our proposal is to replace the averaging prediction of equation (6) by a more precise one, building on the information contained in the available mesh geometry. Changing the prediction is equivalent to choosing a different version of the discrete Laplace operator. A natural choice is a version that produces a zero residual for flat surfaces. Such Laplacian operators are said to have the so-called *linear reconstruction property*. This property does not determine the Laplacian fully, and the literature offers multiple choices. We did experiments with the Cotan Laplacian [PJP93] and with the Mean Value (MV) Laplacian [Flo03].

Both of these choices can be expressed by replacing eq. 6 by a weighted generalization:

$$v_i' = \frac{1}{\sum_{j \in N(i)} w_{ij}} \sum_{j \in N(i)} w_{ij} v_j, \tag{7}$$

where w_{ij} is computed from the geometry of the mesh surrounding the i-th vertex (see figure 5) as

$$\begin{aligned} w_{ij}^{cotan} &= cot(\gamma) + cot(\theta), \\ w_{ij}^{MV} &= \frac{tan(\alpha/2) + tan(\beta/2)}{r_{ij}}, \end{aligned} \tag{8}$$

where r_{ij} is the distance between v_i and v_j.

Note that the neighbourhood relations of the Laplacian should be built from the texture connectivity rather than mesh connectivity. We again assume, that the texture connectivity can only differ from the mesh connectivity in that crease edges are introduced. For each texture triangle, a corresponding mesh triangle always exists, and thus it is possi-

				texture			Weighted	Laplacian		
model	triangles	vertices	parameterization	vertices	RMSE	Parallelogram	prediction	Tutte	Cotan	MV
Horse	96966	48485	ABF [SdS01]	52345	0.0015	2.77	2.59	8.47	**2.00**	2.12
					0.0001	6.33	**2.95**	16.57	4.87	6.30
			ABF++ [SLMB05]	52345	0.0015	2.80	2.53	9.05	**2.01**	2.13
					0.0001	6.37	**2.99**	16.63	4.83	6.25
			DPBF [SdS01]	52345	0.0015	2.77	**2.55**	8.05	2.87	2.63
					0.0001	6.45	**3.23**	16.70	9.66	10.13
			LSCM [LPRM02]	53460	0.0015	2.86	**2.63**	9.30	2.69	2.63
					0.0001	6.48	**2.99**	18.04	6.02	7.33
			HLSCM [RL03]	53460	0.0015	2.82	**2.61**	9.31	2.70	2.63
					0.0001	6.48	**2.99**	18.05	5.42	7.32
Fiery	119344	59727	ABF [SdS01]	66216	0.0030	2.61	2.62	4.12	1.82	**1.79**
					0.0001	7.61	**3.07**	13.22	7.97	6.44
Victoria	32794	17259	manual	19763	0.0008	7.47	**4.83**	12.48	9.63	9.58
					0.0001	13.74	**9.49**	19.84	16.28	15.73
Bimba	17710	8857	ABF [SdS01]	9285	0.0015	4.59	3.07	7.21	1.56	**1.24**
					0.0001	12.68	**5.64**	15.87	7.94	7.19
Frog	40244	20834	manual	21762	0.0015	4.56	**4.03**	8.79	7.64	6.86
					0.0001	10.29	**8.96**	17.43	15.55	14.64
Bunny	51414	29570	unknown	29609	0.0030	3.50	**3.00**	6.62	3.22	3.21
					0.0001	12.52	**6.58**	17.33	12.77	12.17
Kachel	582428	292053	LSCM [LPRM02]	294051	0.0004	3.26	2.82	7.37	**0.27**	0.29
					0.0001	8.80	2.84	12.78	**0.36**	0.39

Table 1: *Data rates for different models and different error levels. All data rates are in bits per vertex, for texture geometry only, i.e. the cost of encoding texture connectivity is not included.*

ble to evaluate the corresponding angles and/or edge lengths required for the construction of the geometric Laplacian.

Note that in order to follow the prediction/correction scenario, we normalize each row of the Laplacian matrix so that the value of all elements on the diagonal is -1. Doing that allows us to quantize the delta coordinates uniformly, since they are equivalent to prediction residuals.

The whole process is performed in following steps:

1. mesh connectivity and geometry is transmitted
2. texture connectivity is transmitted
3. geometric Laplacian L is constructed at both the encoder and the decoder from texture connectivity and mesh geometry.
4. additional anchor points are added to form the extended invertible Laplacian L^*.
5. coordinates of texture vertices are stacked into vectors $\mathbf{u}$ and $\mathbf{v}$.
6. residuals $\mathbf{r_u} = L^*\mathbf{u}$ and $\mathbf{r_v} = L^*\mathbf{v}$ are evaluated, quantized and transmitted to the decoder.
7. The decoder reconstructs the texture coordinates by inverting the process, i.e. $\widehat{\mathbf{u}} = L^{*-1}\mathbf{r_u}$ and $\widehat{\mathbf{v}} = L^{*-1}\mathbf{r_v}$.

Note that here and in the following, a value with a hat denotes a distorted version of the corresponding value, as known to the decoder.

6. Results

We have measured the required data rates and introduced distortions for several textured models. The distortion of the texture coordinates has been measured in terms of the standard Root Mean Squared Error (RMSE), i.e. having original texture coordinates $\mathbf{x}_i, i = 0..V-1$, where V is the number of texture vertices, and their corresponding distorted coordinates $\widehat{\mathbf{x}}_i, i = 0..V-1$, the RMSE is computed as

$$RMSE = \sqrt{\frac{1}{V}\sum_{i=0}^{V-1} \|\mathbf{x}_i - \widehat{\mathbf{x}}_i\|^2}. \quad (9)$$

In the experiments, we have used exact angles from the original mesh geometry, assuming that it has been encoded losslessly. In case of loss of precision in mesh geometry, it is expected that the performance of our proposed coders will decrease slightly, depending on the amount of distortion introduced.

For the experiments, we have used several parameterized models, representing the usual applications. We have used several models with existing parameterization created by artists ("Victoria 4.2" and "Frog" models from DAZ Studio 4.5). Apart from that, we have created parameterizations for

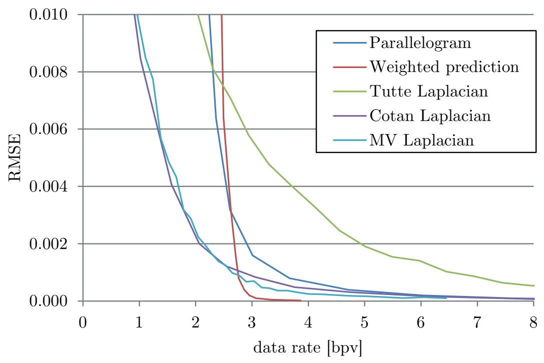


Figure 6: *Rate-distortion chart for the DAZ Studio Fiery scene.*

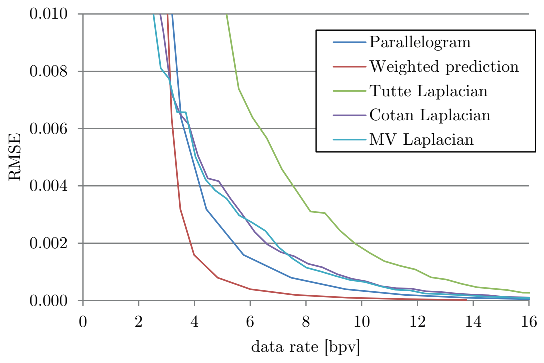


Figure 7: *Rate-distortion chart for the Victoria model.*

other models using the *graphite*[†] tool, which offers several methods of parameterization computation.

Figures 6 and 7 show the typical results obtained in most cases. The Laplacian based encodings generally provide better performance in the area of very low data rates, while the traversal based approaches are the better choice for higher data rates and higher accuracy of reconstruction. The proposed modification of parallelogram prediction using the weighted predictor leads to substantial savings in terms of data rate required for a given precision. The saving may reach up to 60% of the data rate at RMSE of 10^{-4}.

Using a geometrical Laplacian also provides a substantial improvement of compression performance over the use of Tutte (combinatorial) Laplacian. In our experiments, we did not observe any significant difference between the Cotan and MV Laplacian, both of these discretizations provide results that are about 50% better than a combinatorial Laplacian based encoding in terms of reduced data rate at the same reconstruction error.

† http://alice.loria.fr/index.php/software/3-platform/22-graphite.html

Table 1 summarizes the results for different datasets and parameterizations. We have obtained an improvement of compression performance for every parameterization method that we have tested. Generally, however, the improvement is smaller for parameterizations created manually by artists, probably because of limited conformality in such cases. Figure 8 illustrates the effect of reducing the distortion visually.

The table 2 documents the running times for tested compression methods. Note that the key factor influencing the running time of Laplacian based encoding is the searching for optimal anchor point positions, where a large sparse linear system is solved *for each anchor*. We use 50 anchor points additional to those required for matrix regularity. The time required for decoding in Laplacian based coders does not significantly depend on the number of used anchors, because the system is only solved once.

We have experimented with distributing the anchor points randomly, however, the results were not satisfactory. Figure 9 documents the dependency of the performance of Laplacian based encoding on the number of anchor points. It follows from the nature of Laplacian based encoding that some areas may become dislocated in the reconstruction by a large offset, and the optimal anchor placement succeeds in locating these particular areas and correcting them. Trying to achieve the same result by using a larger number of random anchors leads to encoding inefficiency, since the coordinates of anchor points cannot be predicted. Note that the chaotic behaviour of the dependency for low number of anchors is caused by the fact that Laplacian encoding does not involve any means of avoiding error accumulation, except for the anchor points. Therefore sometimes, due to random factors, the accumulated error in areas far away from anchor points cancels out, while other times it does not.

Generally, using a geometric Laplacian instead of combinatorial does not lead to any significant slowdown, neither on the encoder side nor on the decoder side. Using of weighted predictor instead of parallelogram usually leads to a slight slowdown on both sides of the process, on average we have obtained a slowdown of 4% at the encoder side and 15% at the decoder side. Since the processing times of the traversal based coders are generally very short, we believe that the gained performance is worth the increase in processing time. For example, the decoding of the Fiery dataset is about 36ms slower with the weighted prediction, while about 436kB of data is saved. That means that only with a bandwidth of more than 12MB/s is it faster to transmit the data using parallelogram prediction.

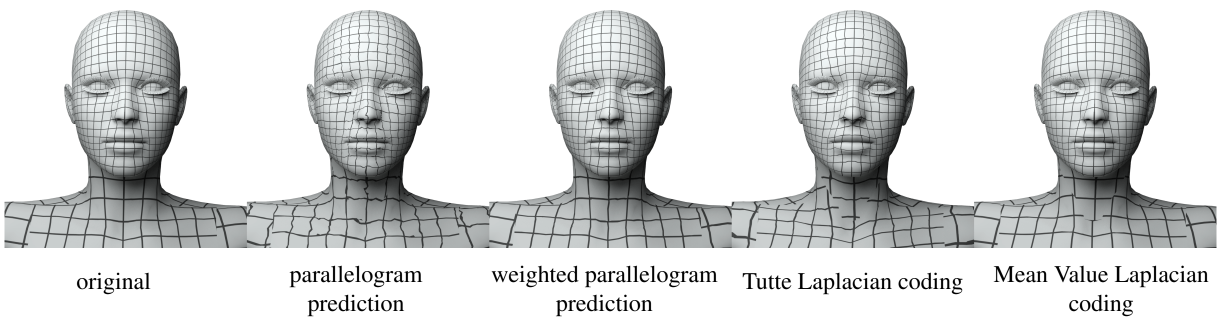

original
parallelogram prediction
weighted parallelogram prediction
Tutte Laplacian coding
Mean Value Laplacian coding

Figure 8: *Visual coparison on the* Victoria *model. All of the distorted versions were reconstructed at a data rate of 5.25 bpv. Notice the different character of the distortion: while the parallelogram based approaches produce a small, noisy local error, the Laplacian based approaches generally produce a smoother error that can be easily spotted on the borders of texture patches (see the neck of the figure).*

			Horse	Fiery	Victoria	Bimba	Frog	Bunny	Kachel
Parallelogram		enc.	334	451	144	78	151	129	2681
		dec.	175	246	81	39	81	67	1631
Weighted		enc.	388	484	153	67	167	129	2633
		dec.	220	278	89	44	98	75	1630
Tutte	50	enc.	29933	39091	8621	5320	14232	9232	253641
	anchors	dec.	1582	2122	527	271	742	488	12277
	100	enc.	57917	75963	16795	10305	25198	17898	497048
	anchors	dec.	1622	2022	502	278	640	480	12792
Cotan	50	enc.	30105	39287	8702	5313	13016	9336	255606
	anchors	dec.	1850	2284	543	309	702	528	12963
	100	enc.	58017	76212	16889	10396	25268	17996	497520
	anchors	dec.	1713	2343	558	309	724	518	13088
MV	50	enc.	29852	39262	8677	5320	13054	9350	254624
	anchors	dec.	1750	2206	558	309	748	516	13260
	100	enc.	57954	76044	16761	10333	25238	18174	496455
	anchors	dec.	1738	2200	574	306	742	518	13041

Table 2: *Processing times for encoding and decoding using different methods. All times are in milliseconds.*

7. Conclusions

We have presented two prediction algorithms designed specifically for compression of texture coordinates, exploiting the information about mesh geometry which is assumed to be available at the decoder side. The two methods are based on the assumed conformality, which is at least partially inherent to most parameterization methods as well as to manually created parameterizations. Our results demonstrate, that in practical cases of both manual and automatic parameterizations, using the proposed algorithms leads to a saving of more than 50% of the required data rate compared with using direct extensions of mesh compression methods such as parallelogram prediction.

The Laplacian based encoding is generally better for very low bitrates, but it suffers from higher computational costs, in particular on the encoding side, while the decoding times remain within the limits of practical applicability for moderately large meshes. The proposed improvements work only if the parameterization is indeed at least partially conformal and continuous, yet conformality is the objective of most automatic parameterization methods and it also corresponds with the objectives artists follow when creating a parameterization manually. Of course, in the extreme case when each geometry triangle is mapped to an isolated texture triangle brings our proposal no improvement at all. Also, for more regular meshes the prediction stencils get closer to the parallelogram shape, which leads to a smaller improvement by using a weighted prediction and thus smaller performance gain. The improvement gained by using a geometric Laplacian instead of a combinatorial Laplacian also gets smaller in such case, since the two become more similar.

We have also briefly revisited the problem of encoding of texture connectivity, providing an analysis that shows that the simple scheme of encoding one bit per edge outperforms the vertex-bit based algorithms.

All of the proposed lossy compression schemes may lead to triangle flip-overs in case of very low bitrates. As future work, we would like to add removal of such artifacts by lo-

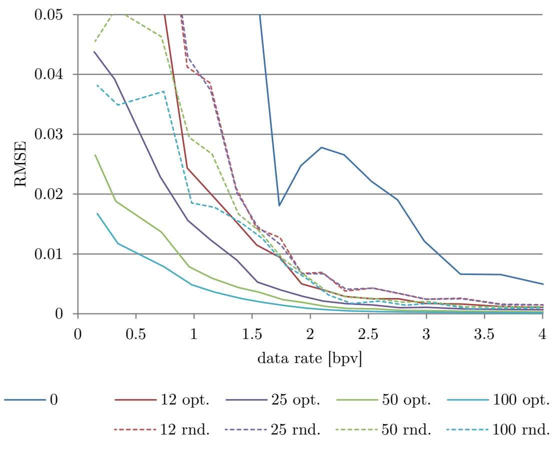


Figure 9: *Performance of MV Laplacian based encoding in dependence on the number of anchors, using the MV Laplacian to encode the Fiery model. Similar results were obtained for other datasets as well.*

cally recomputing the parameterization using the mean value mesh Laplacian, which guarantees an overlap-free mapping. More generally, we would like to test different methods of evaluating the distortion caused by compression of texture coordinates. In particular, we would like to apply perceptual metrics on the images rendered using the distorted texture coordinates. We assume that the results may get different, especially regarding the relative performance of traversal and Laplacian based encoding.

We would also like to test the proposed methods for the task of encoding other vertex- or corner- based mesh properties, such as normals, colors and others. We assume that the proposed improvements will bring a performance gain in such cases as well, rigorous tests however have to be performed in order to confirm such assumption.

References

[CCOST05] Chen D., Cohen-Or D., Sorkine O., Toledo S.: Algebraic analysis of high-pass quantization. *ACM Trans. Graph. 24*, 4 (Oct. 2005), 1259–1282. 6

[Flo03] Floater M. S.: Mean value coordinates. *Comput. Aided Geom. Des. 20*, 1 (Mar. 2003), 19–27. 6

[GS98] Gumhold S., Strasser W.: Real time compression of triangle mesh connectivity. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1998), SIGGRAPH '98, ACM, pp. 133–140. 3, 4

[HLS07] Hormann K., Lévy B., Sheffer A.: Mesh parameterization: Theory and practice video files associated with this course are available from the citation page. In *ACM SIGGRAPH 2007 Courses* (New York, NY, USA, 2007), SIGGRAPH '07, ACM. 4

[Huf52] Huffman D.: A method for the construction of minimum-redundancy codes. *PIRE 40*, 9 (Sept. 1952), 1098–1101. 2

[IS00] Isenburg M., Snoeyink J.: Face fixer: Compressing polygon meshes with properties. 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. 263–270. 3, 4

[IS03] Isenburg M., Snoeyink J.: Compressing texture coordinates with selective linear predictions. In *Computer Graphics International* (2003), IEEE Computer Society, pp. 126–133. 3

[KPRW05] Kälberer F., Polthier K., Reitebuch U., Wardetzky M.: Freelence - coding with free valences. *Computer Graphics Forum 24*, 3 (2005), 469–478. 2

[LPRM02] Lévy B., Petitjean S., Ray N., Maillot J.: Least squares conformal maps for automatic texture atlas generation. *ACM Trans. Graph. 21*, 3 (July 2002), 362–371. 7

[MWS03] Marpe D., Wiegand T., Schwarz H.: Context-based adaptive binary arithmetic coding in the h.264/avc video compression standard. *IEEE Trans. Circuits Syst. Video Techn. 13*, 7 (2003), 620–636. 2

[PJP93] Pinkall U., Juni S. D., Polthier K.: Computing discrete minimal surfaces and their conjugates. *Experimental Mathematics 2* (1993), 15–36. 6

[RL03] Ray N., Levy B.: Hierarchical least squares conformal map. In *Proceedings of the 11th Pacific Conference on Computer Graphics and Applications* (Washington, DC, USA, 2003), PG '03, IEEE Computer Society, pp. 263–. 7

[Ros99] Rossignac J.: Edgebreaker: Connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics 5* (1999), 47–61. 2

[SCO01] Sorkine O., Cohen-Or D.: Warped textures for uv-mapping encoding. In *Proceedings of EUROGRAPHICS, short papers volume* (2001). 3

[SCOT03] Sorkine O., Cohen-Or D., Toledo S.: High-pass quantization for mesh encoding. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing* (Aire-la-Ville, Switzerland, Switzerland, 2003), SGP '03, Eurographics Association, pp. 42–51. 3, 6

[SdS01] Sheffer A., de Sturler E.: Parameterization of faceted surfaces for meshing using angle-based flattening. *Eng. Comput. (Lond.) 17*, 3 (2001), 326–337. 7

[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. 4

[SLMB05] Sheffer A., Lévy B., Mogilnitsky M., Bogomyakov A.: Abf++: Fast and robust angle based flattening. *ACM Trans. Graph. 24*, 2 (Apr. 2005), 311–330. 7

[TG98] Touma C., Gotsman C.: Triangle mesh compression. In *Graphics Interface* (June 1998), pp. 26–34. 2

[THLR98] Taubin G., Horn W., Lazarus F., Rossignac J.: Geometry coding and vrml, 1998. 3

[TR98] Taubin G., Rossignac J.: Geometric compression through topological surgery. *ACM TRANSACTIONS ON GRAPHICS 17* (1998), 84–115. 2

[VB13] Váša L., Brunnett G.: Exploiting connectivity to improve the tangential part of geometry prediction. *IEEE Trans. Vis. Comput. Graph. 19*, 9 (2013), 1467–1475. 3, 5