EUROGRAPHICS 2005 / M. Alexa and J. Marks
(Guest Editors)

Volume 24 (2005), Number 3

FreeLence - Coding with Free Valences

Felix Kälberer[†] Konrad Polthier[†] Ulrich Reitebuch[†] Max Wardetzky[†]

Zuse Institute Berlin

Abstract

We introduce FreeLence, a novel and simple single-rate compression coder for triangle manifold meshes. Our method uses free valences and exploits geometric information for connectivity encoding. Furthermore, we introduce a novel linear prediction scheme for geometry compression of 3D meshes. Together, these approaches yield a significant entropy reduction for mesh encoding with an average of 20-30% over leading single-rate region-growing coders, both for connectivity and geometry.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Surface mesh compression, connectivity coding, geometry coding.

1. Introduction

Compression of digital geometry models is the answer to an industrial demand: Ever-finer geometric detail, requiring millions of vertices, is part of the everyday agenda in the movie industry, the computer aided design (CAD) industry, and in server-sided rendering applications. Over the last years, many exciting ideas and new theoretical insights have been devoted to finding ways of reducing the amount of storage such models absorb. Some of those ideas have become industrial standards, like the compression methods built into MPEG-4 and Java3D. Different requirements gave rise to differing solutions with varying trade-offs between efficiency of representation and accuracy of detail - there are lossless and lossy approaches, there are wavelet and spectral decomposition methods, there are progressive as well as single-resolution techniques. But often, such as for detailed mechanical components in CAD systems, lossy storage is prohibitive, and this is where lossless coders enter. *Lossless* stands for the ability to encode the floating point positions of the mesh at highest accuracy; in practice, positions are often quantized to 10-14 bits per coordinate, a concession which has turned out to be tolerable in applications.

It seems fair to say that among the lossless, single-resolution coders, the TG-coder [TG98], Gumhold's Cut-Border-Machine [GS98], and Rossignac's Edgebreaker [Ros99] stand out not only for their simplicity and robustness combined with competitive compression rates, but also because they have spawned numerous descendants. These schemes have seemingly subtle yet conceptually crucial differences.

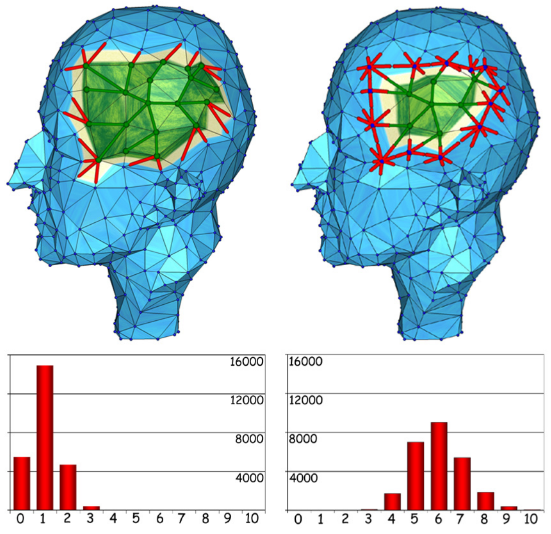

Figure 1: *Geometry-driven coding with free valences (left) will* in practice *yield a lower symbol dispersion than coding with full valences (right). The frequency of local and global splits is negligibly low for this particular model.*

[†] Supported by the DFG Research Center MATHEON "Mathematics for key technologies" and mental images GmbH.

Full-valence coding. The most prominent full-valence (or degree) coder was introduced by Touma & Gotsman [TG98]. Although it has seen slight extensions and improvements, the method has stayed unchanged at its heart. The mesh is conquered by a region-growing traversal in a spiraling manner. During traversal the coder maintains an *active list* that separates the already encoded region from the unencoded part. The combinatorics (or connectivity) of the mesh is encoded by subsequently picking a *focus* vertex on this active list and storing the *full valence* of every *adjacent* unencoded vertex. As at this point no information about the focus itself is stored, the TG-coder could be said to store information *ahead of time*. Only in few cases, when the active list touches upon itself, additional symbols must be spent in order to properly decode this so called *split situation*. One attractive aspect of full-valence coders stems from the fact that their total code length almost matches the Shannon entropy [Sha48] of the involved full valences alone. Tutte [Tut62] provided a theoretical lower bound for the asymptotic length of any triangle mesh code by counting the number of differing rooted planar triangulations for a given number of vertices. Later it was shown in [AD01b] and [KADS02] that the asymptotic worst-case bit rate for full-valence coders is exactly equal to Tutte's census, provided that one counts out split events. It was a break-through by Poulalhon and Schaeffer [PS03] to develop a connectivity coder (not being valence-based) which works precisely at the Tutte limit. In practice, however, their approach consumes a constant number of bits for all meshes with the same number of vertices, at least if used without a context-based entropy coder; and their coder is not easy to generalize to meshes of higher genus.

Label-based coding. The label-based coders *Edgebreaker* and *Cut-Border-Machine* were developed independently by Rossignac [Ros99] and Gumhold & Strasser [GS98]. Like the TG-coder, label-based methods maintain an active list that bounds the processed region. Successively the unencoded triangle incident to a distinguished *gate* edge on the active list is conquered. If the third vertex of that triangle is yet unconquered, Edgebreaker stores a **C** (*create*) label. Otherwise, the third vertex must have been previously encoded, and its incident edges in the new triangle may or may not lie on the active list. The four cases that can thus arise are distinguished by the labels **R** (*right*), **L** (*left*), **S** (*split*), and **E** (*end*). The gate is subsequently moved according to the last symbol. The Cut-Border-Machine and Edgebreaker are almost identical. Their only real difference is the handling of splits.

Labels vs. degrees. There are important conceptual differences between label-based coders and degree coders. Label coding is *triangle-based* - in Edgebreaker, for example, every triangle contributes exactly one symbol. In contrast, degree coding is *vertex based*: degree symbols of the TG-coder are directly corresponding to the vertices of the mesh, and triangles are encoded implicitly without having any explicit symbol that could be said to correspond to them. Moreover, label coders encode mesh information locally, right at the gate of the active list; in contrast, degree coders store information ahead of time and maintain global information about the entire active list. As a consequence, degree coders will never split off regions containing no unprocessed vertices, whereas label-based coders lack the ability to recognize such situations, called *warts* [TR98].

Coding with free valences. Akin to the above coders, FreeLence conquers the vertices of a mesh by a growing disk. Except for split events, coding with free valences stands for storing the number of *unconquered vertices* connected to the focus, and, as in full-valence coders, the number of symbols is closely related to the total number of vertices, so that their code sequences and symbol dispersions can be compared as in Figure 1. Despite this similarity in symbol counts, FreeLence is conceptually a label-based coder and not a degree coder. FreeLence encodes information about vertices right at the active contour, and the resulting code sequence can be translated directly into an extended Cut-Border-Machine code, see Section 2.5. Similar to Edgebreaker and the Cut-Border-Machine, switching from full valences to free valences introduces warts which are avoided entirely in full-valences coders. To reduce the impact of such unwanted events, FreeLence introduces two new *local split* symbols, **P** (previous) and **N** (next). The separation of concepts between general splits and local splits is essential for the success of our coder in practice.

Geometry-driven mesh traversal. The concept of using free valences alone, without an appropriate mesh traversal, will not outperform existing connectivity coders. FreeLence employs a *geometry-driven traversal scheme* as in [LAD02], which keeps the active boundary locally as convex as possible. The basic idea is to pick the next focus according to the minimal *opening angle* along the active contour. On the one hand, this approach significantly reduces symbol dispersion in the sequence of free valences. On the other hand, occurrence of costly split symbols, which destroy code coherence, is reduced. In addition, FreeLence uses opening angles as *contexts* for storing free valences. The use of contexts results in a further significant drop of bit rates.

Geometry encoding. To encode the geometry part of the mesh, each vertex is stored as an offset from an educated guess of its (quantized) position. The beautifully simple *parallelogram rule* [TG98] makes such a guess by assuming that two adjacent triangles form a parallelogram. This has proved to be a powerful choice in practice. The parallelogram rule has seen improvements and extensions: Isenburg and Alliez [IA02] generalize it to polygon meshes, Kronrod and Gotsman [KG02] propose a spanning tree to optimize for the best-suitable parallelograms and Cohen-Or et al. [COCI02] use several parallelogram predictions simultaneously. As observed in [LAD02] and [GA03], it is highly beneficial to move to *local coordinate systems*, defined by one or several *contributing triangles*. FreeLence takes up the scheme of quantizing in local coordinates and augments it by a novel *local linear prediction rule*. In essence, by extending the stencil of vertices which contribute to prediction, we observe significantly lower bit-rates while keeping the same L^2-error as other single-rate region-growing coders. Furthermore, we show how our particular choice of weights can be extended to an entire space of *reasonable local weights*. We report first evidence that separate model classes such as models produced by CAD applications as well as very regular models like subdivision surfaces on the one hand, and very irregular models on the other, tend to form *clusters* in this space of local prediction weights.

Contributions. *Connectivity coding* based on free valences as a new variant of label-based coding. Free valence codes highly benefit from a geometry-guided mesh traversal which significantly reduces symbol dispersion in the connectivity code. This scheme is augmented by the concept of encoding free valences based on *angle contexts* on the one hand, and the introduction of *local split labels* on the other.

Geometry prediction in local coordinates based on a simple multi-gate parallelogram approach, significantly reducing bit-rates compared to the classical parallelogram rule. This new scheme embeds into a larger space of linear predictions. First evidence suggests that different 3D model classes tend to form distinct clusters in this larger space.

Together these approaches improve existing single-rate connectivity coders by 20-30%.

Other relevant work. The literature on digital geometry compression has been vast over the last few years and is rapidly growing. This is not the place to give an exhaustive account of it; for an excellent survey we refer to Alliez and Gotsman [AG03]. Here we give a short overview of those concepts which are relevant to our work. Descendants of the pioneering approaches to single-rate compression are numerous - such as the improved Cut-Border-Machine [Gum99], improvements of the gate-based Edgebreaker as in [KR99], [RS99], [IS99], [IS00], [KG00b], [Szy02], [CR04], as well as extensions of valence-based techniques as in [AD01a], [AD01b], [Ise02], [KG02]. The interplay between geometry and connectivity has been investigated by several authors. The approach of [IGG01] shows that a lot of geometric information is contained in the connectivity of a mesh. But since geometric information is also completely contained in the vertex positions which are saved along with the connectivity, some of this information is redundant. There are two natural ways to remove this redundancy: Using connectivity information to improve geometry coding or using geometry information to improve connectivity coding. The first approach was investigated in a series of articles by Gotsman et al. [KG00a], [KG01] and [BCG05] who have studied spectral decompositions of the discrete graph Laplacian for geometry encoding. The second approach was for example taken up in [GD02] and [LAD02]. Our approach, too, fits more into this latter category.

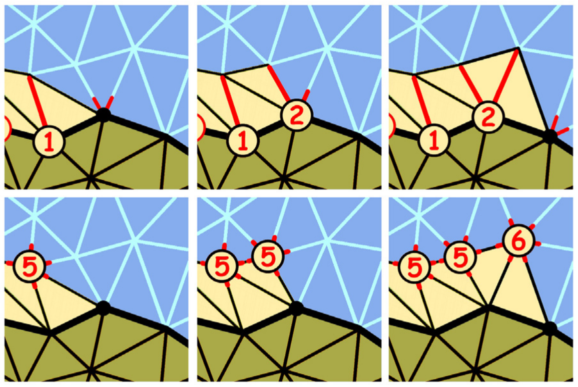

Figure 2: *Encoding with free valences (top) versus encoding with full valences (bottom) along the active contour.*

2. Coding with free valences

In this section we explain how FreeLence encodes connectivity of 2-manifold triangle meshes (with or without boundary) by working with free valences. We first review some relevant terminology and concepts; in particular it is important to clearly distinguish between the notions of *free vertex*, *free edge* and *free valence*.

Free Vertex: A vertex of the mesh which has not been previously visited by the encoder.

Free Edge: An edge not previously conquered.

Active List (or cut border): A boundary component of a fully conquered sub-complex of the mesh, separating the mesh locally in an inner region of conquered edges and an outer part containing free edges. Due to global splits, there may be several active lists at a time.

Focus (or pivot): The vertex in an active list which is currently processed by the encoder or decoder. How the focus is chosen depends on the *traversal scheme*.

Split: A split event occurs if a *free edge* connects the focus with a previously *conquered vertex* on the same active list.

Full Valence: The total number of edges incident to a vertex.

Free Valence: In general, the number of *free vertices* connected to the focus by an edge, except at split vertices where the situation is more subtle, see Section 2.4.

Opening Angle: Assigned to each vertex in an active list.

$$\textit{opening angle} = 2\pi - \sum_i \angle(e_i, e_{i+1}),$$

the sum being taken over all angles between adjacent *conquered edges* incident to the vertex.

Encoding: Akin to existing popular methods we employ a region-growing strategy to conquer the mesh. Throughout the encoding, we maintain one or more lists of vertices, the *active lists*, which constitute the boundary of the so far processed region. Initially the link of an arbitrary vertex will serve as the first active list. The vertex with minimal opening angle is chosen as the focus. Unless a split situation is encountered, the number of free valences of the focus is stored. The free vertices around the focus are conquered in clockwise order and their positions are encoded, as described in Section 3. The active list is updated by replacing the focus by the newly conquered vertices. Updating opening angles and picking a new focus finishes the iteration. The algorithm is explained in more detail in Sections 2.1 and 2.4.

2.1. Separation of concepts - local versus global splits

When the current focus has a free edge connecting it to a previously conquered vertex on the same active list, the active list needs to get split along the connecting edge. In order to properly decode this situation, a special split symbol together with a unique identifier to restore the target vertex is stored (usually as an offset from the focus). Full-valence coders such as the TG coder will only split-off active lists which enclose at least a single free vertex as they know *all free edges along the entire active list* and can therefore close triangles with zero free edges early. The situation changes for a free-valence coder, where only the information about the *current focus* is fully available. As a consequence, free-valence coders may split off regions containing no free vertices in their interior. We observe that the vast majority of such situations happens if the next or previous vertex to the focus contains zero free edges. To account for this fact, we introduce two new coding symbols - **N** (next) and **P** (previous). We call them **local splits**, compare Figure 3; all other splits are called **global splits**.

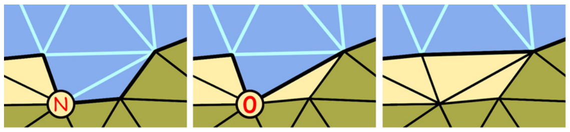

Figure 3: ***A local split*** *occurs if a neighbor of the focus has zero free edges. The sequence shows how to avoid wasting a full split: FreeLence codes a next symbol* **N**, *closes the next vertex, and proceeds at the focus.*

2.2. Geometry-driven traversal

We observe that bit rates for storing free valences significantly depend on a clever traversal scheme. FreeLence *exploits geometric information* about the mesh by picking the next focus according to the *smallest opening angle* along the active contour. This geometry-guided traversal is highly beneficial for optimizing free valence codes in two ways:

Symbol dispersion. Ignoring global splits, each vertex gets assigned exactly one of the symbols **P**, **N**, **0**, **1**, **2**, ...

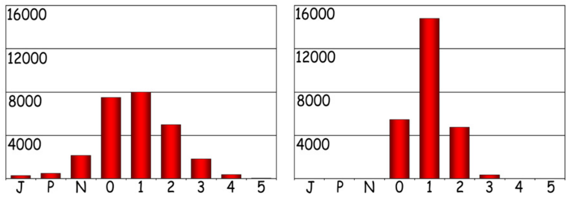

Figure 4: *Symbol frequencies of free-valence codes of the* max *model for a simple spiraling mesh traversal (left) vs. FreeLence's geometry-driven traversal (right). Local splits are denoted by* **P** *(previous) and* **N** *(next), global splits and merges by* **J** *(join).*

If V denotes the total number of vertices, then, assuming no global splits,

$$V = V_P + V_N + V_0 + V_1 + V_2 + \cdots,$$

where V_P stands for the number of **P**-vertices, V_N stands for the number of **N**-vertices, and V_i denotes the number of vertices with i free valences. Moreover, ignoring global splits, during traversal of a free valence coder, each vertex is *free exactly once*, i.e. there exists exactly one focus for which a vertex is counted as a free vertex. Hence, $V = \sum_1^\infty i \cdot V_i$, so that

$$V = 0 \cdot (V_P + V_N + V_0) + 1 \cdot V_1 + 2 \cdot V_2 + \cdots$$

Combining these equations shows that the penalty for each free valence i, with $i > 1$, are $(i-1)$ symbols out of **0**, **P**, **N**. At the same time, our traversal scheme, by choosing smallest opening angles, tends to penalize large free valences, and the number of **0**'s, **P**'s, and **N**'s is thus also reduced. Therefore, geometry-guided traversal yields *in practice* symbol dispersions concentrating around the *average free valence* 1 as shown in Figure 4.

Split reduction. By avoiding local peeks and pokes in the active list, not only the occurrence of local splits (**P** and **N**) is reduced, but also the frequency of splits in general is kept at low levels, see Figure 4.

2.3. Context-based coding - exploiting regularity

Full-valence coders, by design, adapt to combinatorially regular meshes, other schemes like Angle-Analyzer [LAD02] do not. Free-valence coders perform well for highly regular meshes if used with a suitable *context-based entropy coder*. To achieve adaption to regularity, FreeLence stores free valences in separate tables (contexts) which are chosen according to the opening angle of the current focus, compare Figure 5. This context-based encoding relies on the following simple heuristics: *small angles usually indicate a smaller number of free edges, and bigger angles a bigger number*. In our implementation we are using a greedy approach to further optimize the angle contexts according to their *actual distribution* of the respective mesh in such a way, that the sum over all entropies in the tables becomes minimal. The

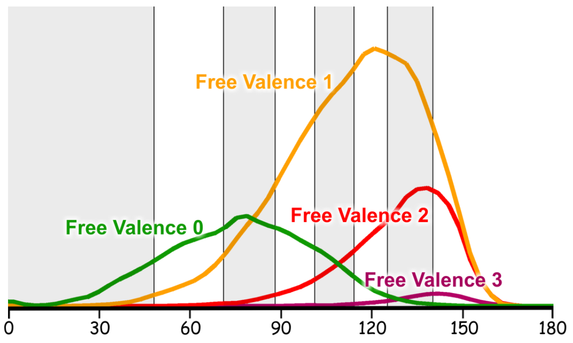

Figure 5: *Distribution of opening angles for an encoding run of the* max *model. Each free valence has its own distribution. Note how the curves for different values of free valences overlap due to the irregularity of the mesh. Negative opening angles can occur at vertices with negative Gauß curvature and at boundary vertices. The vertical gray and white bars show the optimized thresholds by which FreeLence groups symbols into contexts.*

vertical lines in Figure 5 show such an optimized dissection for the *max* mesh. Each table is compressed separately with an order-0 entropy coder [WNC87]. By using angle contexts, we observe a gain of about another 25% for the entropy of the connectivity code. Note that angle-based context modeling can not be applied to full-valence coders such as [TG98] and [AD01b] because degree coders store no information about the current focus, but only information about adjacent vertices, the degrees of which are not correlated to the opening angle at the focus. On the other hand, label coders may very well benefit from angle contexts.

2.4. Coding details

FreeLence operates on two main data structures - a single *priority queue* where all active vertices are sorted by their opening angles, and one or more doubly linked lists of these vertices, the *active lists*. The algorithm uses a finite set of *labels*, **P** (previous), **N** (next), **J** (join), **D** (dummy), and a set of *numbers* which specify *free valence* and *offset*. The algorithm works as a finite state machine. Special symbols like *previous*, *next*, *join*, and *dummy* are represented as negative integers -1, -2, -3 and -4 to distinguish them from free valences. The algorithm starts by picking an arbitrary seed vertex v, the first *focus*, and its full valence is written out. Then the star of v is closed in clockwise order, and the link of v constitutes the first active list. At every update, an *opening angle* is assigned to each vertex in the active list, and a new focus is repeatedly chosen according to the minimal opening angle. As all vertices in active lists reside in a single priority queue, the focus may jump between different lists. If there is no split, the number of free valences of the focus is stored, and its incident triangles are considered *processed*. The focus is removed from the active list and the priority queue, and the newly conquered vertices are inserted. This scheme is repeated until a focus is hit which has a free edge connecting it to a previously visited vertex; in this case, a special *split* symbol needs to be stored.

Splits and merges. A *local split* occurs if the successor or predecessor to the current focus has zero free edges, and we store the symbol **N** resp. **P** in that case. The next or previous vertex can then be conquered without storing an additional number. After local splits have been processed, we proceed conquering the free edges of the focus in clockwise order. If a free edge is hit connecting the focus to a vertex on some active list, then a *join* event occurs. Joins either correspond to global splits, i.e. connect the focus to a vertex in the same active list or to *merges* connecting it to a different active list. The number of merges equals the genus of the surface. When a join is encountered, a label **J** together with the number of free edges in clockwise order to the join edge is stored. A second identifier is needed by the decoder to uniquely locate the target vertex in the queue. Commonly, this identifier is defined by the offset of the split vertex to focus in the active list. However, in order to avoid the need to distinguish between splits and merges, we order the vertices in the priority queue according to their Euclidian distance from the focus. The position of the join vertex in this ordering is used as the missing unique identifier. This identifier is usually a much smaller number than the commonly used offset identifier within the active list. It is stored in a separate *offset list*. After a join has been processed, the active lists and the priority queue are updated and a new focus is chosen.

Boundaries are handled similar to the TG coder. The vertices of each boundary component are connected to a new dummy-vertex, so that the mesh becomes closed. When the encoder hits a dummy-vertex for the first time, a special boundary code **D** is written to provide a hint for the decoder to remove it upon decoding. Since the dummy vertex has no geometric position, we define its incident angles to be $\pi/2$.

2.5. Analysis and Discussion

Using free valences together with a geometry-driven mesh traversal significantly better compresses mesh connectivity in practice than previous single-rate coders, cp. Table 1.

Free valences, degrees, and labels. FreeLence symbols can be directly translated into an extended Cut-Border-Machine code: A number of i free valences corresponds to a sequence of i consecutive **C** labels followed by a single **R** label. Similarly, the FreeLence label **P** corresponds to the Edgebreaker symbol **L** (called *connect backward* in the Cut-Border-Machine). The **N** label has no correspondence in Edgebreaker or the Cut-Border-Machine, but this symbol could be additionally integrated into these coders to treat special warts. Splits can be handled similarly to the Cut-Border-Machine. Although it is possible to translate free-valence codes into label sequences, the translated label code

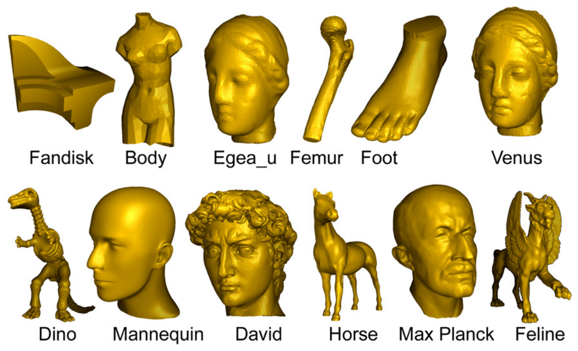

Figure 6: *Benchmark testing models.*

will not benefit as strongly as the original free valence code from geometry-guided traversal if used with an order-0 entropy coder. In fact, the number of **C** labels in the translation corresponds to the sum of free valences, which is the number of vertices of the mesh. Moreover, the *total* number of translated labels roughly equals the number of triangles (about twice the number of vertices), and about 90% of all labels different from **C** are **R** labels in practice. Hence, the number of both **C**'s and **R**'s is roughly equal to the number of vertices. Any rearrangement of them, due to a different traversal, will therefore not affect compression rates if used with an order-0 entropy coder as in FreeLence because the code length of an order-0 entropy coder only depends on the *frequencies* of input symbols. Similarly, the compressed length of full valence sequences is unaffected by the particular choice of a traversal scheme since degree coders store the full degree of every vertex exactly once.

Warts are situations of global splits which do not occur for full-valence coders: In the case where an active list is split off containing more than three vertices but no interior free vertex - similar to the situation in Edgebreaker and the Cut-Border-Machine. For an extensive discussion of warts we refer to Isenburg and Snoeyink [IS04]. In practice, we remove the majority of warts by introducing the symbols **N** (next) and **P** (previous). Note that one could, in theory, avoid warts entirely by *locally remeshing*. In fact, one could even avoid local splits (Figure 3) by applying a single edge flip (corresponding to writing a 0 at the focus instead of **N** or **P**). Yet, edge flips are lossy so that we did not further pursue this direction here. In summary, although warts result in a greater number of possible global splits, by employing our geometry-driven traversal, we have not observed any drastic such effects in practice.

Influence of quantization. Since we are using geometric properties of the mesh (the opening angle) to improve compression of the combinatorics, our connectivity rates depend on the quantization of the geometry. Lower precision results in an increasing number of splits and a reduction of the positive effect of angle contexts, and hence can cause a drop of our rates. However, the correlation between quantization and connectivity bit rates is *reasonable*. For the models listed in Table 1, 10 bit global quantization only yields an average drop of 6% compared to 12 bit global quantization, and models showing significant change in bit rates, such as feline, always showed visually noticeable geometric distortion, too. For 14 bit global quantization there is no measurable difference to 12 bit global quantization. We further counteract this effect by quantizing in a *local coordinate systems*, as described in the next section, based on the observation that angles are far less prone to drastic changes if quantized in a local system which adapts to the *tangency of the mesh*.

model	verts	VD bpv	AA bpv	FL bpv	vs VD	vs AA
body	711	2.38	1.96	1.90	20%	3%
femur	3,897	2.71	n/a	2.11	22%	n/a
random	4,338	n/a	0.57	0.36	n/a	37%
egea_u	5,315	1.63	0.81	0.54	67%	33%
fandisk	6,475	1.02	n/a	0.74	27%	n/a
venus	8,268	2.71	1.95	1.73	36%	12%
foot	10,016	2.20	1.56	1.34	39%	14%
sphere	10,242	n/a	0.23	0.03	n/a	85%
manneq.	11,706	0.37	n/a	0.38	−2%	n/a
dino	14,070	2.25	1.69	1.44	36%	15%
tf2	14,169	n/a	1.01	0.60	n/a	40%
horse	19,851	2.25	1.35	0.96	57%	29%
david	24,085	2.52	n/a	1.98	22%	n/a
max	25,445	2.22	1.45	1.19	47%	18%
feline	49,864	2.38	1.50	1.23	48%	18%
average					34%	28%

Table 1: ***Connectivity compression rates.*** *Comparison of FreeLence (FL) based on 12 bit globally quantized geometry with the Valence-Driven coder (VD) by Alliez-Desbrun and Angle-Analyzer (AA) by Lee-Alliez-Desbrun. The numbers are in bits per vertex. The last two columns show the improvements of FL over VD and AA.*

3. Geometry compression

The bulk of information of a 3D mesh is stored in the positions of its vertices rather than the way these vertices are connected. Therefore it is imperative for any mesh compressor to address this issue. Motivated by the FreeLence geometry-driven traversal scheme, we introduce a new and simple geometry compression method which is based on linear vertex weights around the focus.

Global vs. local systems. The classical TG coder quantizes and predicts in a *global coordinate system*, defined by object space. To that end, the object bounding box is regularly *n-bit* quantized into voxels of side length $l_{vox} = l_{max}/2^n$, where l_{max} is the length of the longest edge of the object's bounding box. The vertices of the mesh are then snapped onto their nearest neighbors in the resulting grid. The core problem with global quantization is high *normal distortion*

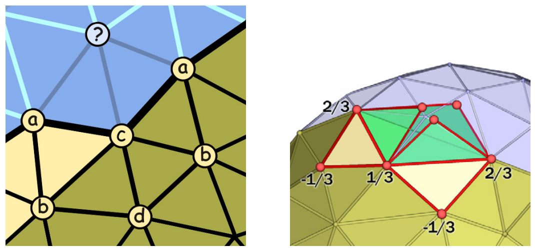

Figure 7: ***Left*** *In a hexagon, the unknown point (labeled "?") is to be predicted from the five known neighbors of the focus (which has weight c). Equal weights are chosen due to invariance under symmetry.* ***Right*** *The FreeLence rule, showing our choice of weights around the focus used for prediction. The rule is used for the last free edge of any focus of degree greater or equal to five.*

of quantized geometry, resulting in a *staircase look*. To avoid this effect it is highly beneficial to move to a *local coordinate system*, defined by one or several *contributing triangles*. This approach was pursued by Lee et al. [LAD02] and Gumhold and Amjoun [GA03]; Gumhold and Amjoun use higher-order (non-linear) prediction, resulting in a high payoff in run time, whereas Lee et al. do not use any prediction but store coordinates in the local system directly.

3.1. Freelence geometry prediction

Choosing local weights. FreeLence adopts the approach of local coordinates and augments it by a simple and powerful *prediction scheme*. The scheme is simple because it only uses *linear weights* of a *local stencil* around the focus. Before discussing a whole space of possible local predictions, we explain the motivation behind the weights we have chosen. First of all we observe that the simple parallelogram rule [TG98] can be improved by employing a bigger stencil. Secondly, we observe that it is beneficial to use *multi-way* parallelogram predictions across both *existing* and *virtual* edges of the mesh. Virtual edges are employed to defeat the given connectivity of a mesh for an irregular layout of its vertices. In FreeLence, we *average* over parallelogram predictions across three edges: the two edges connecting the focus with its direct neighbors on the active list as well as the one virtual edge between the previous and next vertex. This scheme can be used for the last free edge of any focus which has full degree greater or equal to five, see Figure 7. For cases where it cannot be employed, we fall back to the usual parallelogram rule.

Quantization and visual quality. The *offset vector* between the predicted position of the new vertex and its actual position is stored in a *local coordinate system*. This system is determined by choosing a *tangent space* at the focus. Our choice is as follows: The tangent space at the focus is spanned by two vectors - the first being the vector which connects the focus to the predicted vertex and the second being parallel to the line connecting the previous and next vertex on the active list. We then put the x-axis along the line connecting the focus to the predicted vertex and the y-axis orthogonal to it in the tangent space. The z-axis is determined by the cross product between the x- and y-axis. Offset vectors can then be quantized *differently* in tangential and normal direction - based on the observation that tangential drift will be far less visually noticeable than normal drift and the fact that local coordinate systems can separate the typically small *prediction errors* in normal direction from the typically larger prediction errors in tangential direction. To avoid error accumulation during encoding, it is important to snap each encoded vertex to its quantized position. We measure the L^2-error between the quantized mesh and the original using the `Metro` tool [CRS98]. The optimal bit-rates for quantizing separately in tangential and normal direction with L^2-errors matching that of Angle-Analyzer are shown in Table 2. In practice, by turning this observation around, this implies that for quantizations with the same bit-rates as for other coders, we achieve a much better *visual quality* by significantly reducing their L^2-error. Figure 8 shows an example of this effect.

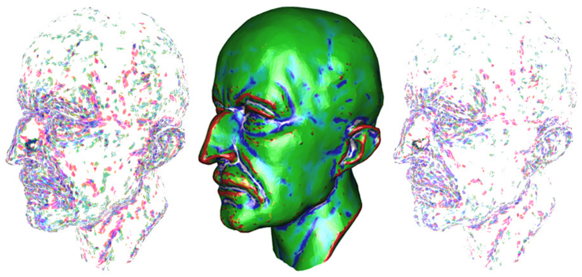

Figure 8: ***Middle*** *Mean curvature plot of the* max *model.* ***Left*** *Difference picture between the mean curvature plots of the original and the globally* 12 *bit quantized model with a geometry bit-rate of* 16.43 *bpv (TG) and a* L^2*-error of* 0.000131. ***Right*** *Difference picture between the mean curvature plots of the original and FreeLence's locally quantized model with a geometry bit-rate of* 15.21 *bpv and a* L^2*-error of* 0.000032.

3.2. Optimality and prediction clusters

We have experimented with a wide variety of local linear weights on a vast variety of digital models in search for a *universally optimal scheme*. It turns out that such a scheme does not seem to exist - in fact, for different model classes the optimal weights tend to form clusters around only a few distinct values. Such clusters arise in the space of *reasonable weights*. We call these patterns **prediction clusters**.

A class of reasonable weights. In a first approach to find optimal weights, we restricted our analysis to situations where the focus has (full) degree *six* and *one* unknown vertex in its link. Symmetry with respect to the line connecting the

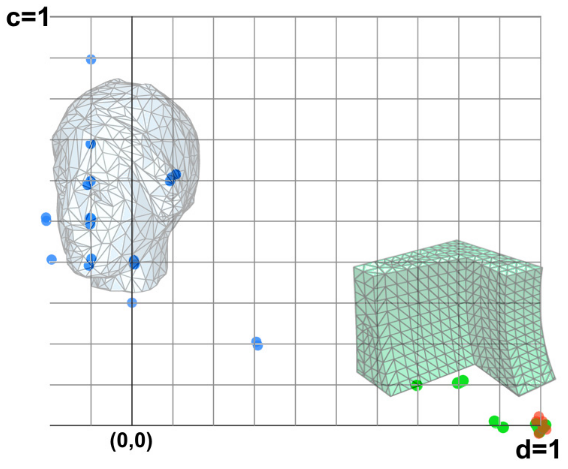

Figure 9: *Different model classes form distinct clusters in the moduli space of prediction weights:* ***green*** *points correspond to subdivision models produced by Loop, Butterfly and $\sqrt{3}$-subdivision,* ***red*** *points correspond to CAD models, and* ***blue*** *points to all other cases.*

focus to the unknown vertex leaves four linear weights to be determined, see Figure 7. We make the following natural assumptions for choosing these weights:

- Affine invariance of the prediction scheme.
- Exact prediction in a regular planar hexagon.

Affine invariance holds if and only if the weights sum to one,

$$2a+2b+c+d=1.$$

Exact prediction for a regular hexagon implies that

$$3a+b+c=2,$$

which can be seen by putting a regular hexagon of edge length one upright into the plane such that the vertex of weight d coincides with the point $(0,0)$ and the unknown vertex lies at the point $(0,2)$. This gives two equations on four unknowns, resulting in a *moduli space of predictions*, forming a 2-dimensional plane for vertices in general positions. If one chooses to parametrize this plane by the unknowns (c,d), then a and b depend linearly on c and d. For example, the FreeLence prediction scheme uses the weights $(c,d)=(1/3,0)$ which has the benefit to work on any focus of degree greater or equal to five.

Prediction clusters. We have searched for lowest bit-rates by determining the optimal (c,d)-prediction parameters separately for each model from a pool of varying digital surfaces. In our tests, we used the new prediction for focuses of degree six; all other cases were treated by the parallelogram rule. We noticed the formation of certain prediction clusters - models of the same class tend to cluster around the same optimal (c,d)-predictions. In particular, very regular models such as those produced by repeated subdivision as well as those produced by CAD applications, cluster around $(c,d)=(0,1)$, which results in the mask $a=d=1$, $b=-1$ and $c=0$. For testing regularity we used common benchmark models, simplified them and applied Loop, Butterfly and $\sqrt{3}$-subdivision. On the other end of the spectrum there are irregular models (in the combinatorial and geometric sense) such as david, body, max, dino, feline, femur, foot and venus which tend to cluster around $(c,d)=(1/2,0)$. Semi-regular models such as bunny and cow can be found in-between. In summary, it seems that the *search for a universal prediction rule is a Sisyphean task*. On the other hand, due to clustering, there is good evidence that certain model classes have specific optimal schemes associated with them.

3.3. Analysis and Discussion

Extending the simple parallelogram rule to a linear prediction scheme on a bigger stencil proves superior to existing single-rate geometry coders in practice. Our new prediction scheme can be applied whenever the degree of a given focus is greater than four and there is exactly one unknown vertex in its link. Hence, the total number of focuses to which our new scheme can be applied *depends on the traversal method*. The FreeLence rule of choosing the next focus according to minimal opening angles prefers situations where the focus has *one free edge*, and is hence the optimal traversal scheme for our prediction. Note how the same angle-based traversal is beneficial both for our connectivity encoding as well as our geometry encoding techniques. The results for geometry compression on benchmark models are summarized in Table 2.

Tangential vs. normal quantization. As in [LAD02], offset vectors are quantized differently in tangential and normal direction. These quantizations are based on the length of the longest edge of the object's bounding box, rescaled differently for tangential and normal quantization. For reproducibility, the optimal rescaling factors which minimize the geometry code length under the constraint to match the L^2-error produced by 12 bit global quantizations are provided in Table 2. Automating the choice of different quantization values in tangential and normal direction is an important topic for future research.

Choosing your weights. Extending the prediction stencil for a linear prediction scheme implies new degrees of freedom for the optimal weights to choose. Some of these degrees of freedom can be eliminated by requiring *natural conditions*, such as affine invariance. We also chose to include the requirement of exact prediction for regular hexagons - motivated by the fact that the average vertex degree in a triangular mesh is equal to six - a consequence of the Euler formula - and the assumption that a prediction scheme should be invariant under symmetries. Yet, these two requirements do not single out a unique choice of weights. The FreeLence

model	AA bpv	FL bpv	AA L^2	FL L^2	tan / nor	vs AA
random	11.20	6.95	102	100	13.4 / 0.67	38%
egea_u	14.90	12.80	39	37	3.7 / 0.48	14%
sphere	4.91	2.19	89	90	15.0 / 0.83	55%
tf2	14.18	11.41	39	41	3.2 / 0.48	20%
horse	12.68	10.35	215	44	4.0 / 0.67	18%
max	11.93	9.41	127	129	7.1 / 0.63	21%
feline	12.84	9.93	52	55	3.7 / 0.63	23%
average						27%

Table 2: ***Geometry compression rates.*** *Comparison of bit rates and L^2-errors (in units of 10^{-6}) between Angle-Analyzer (AA) and FreeLence (FL). AA reported separate rates for angle space and local space; for each model we compare to their lowest value. L^2-errors are produced with the `METRO` tool with 100,000 samples per area unit and vertex and edge sampling turned off; they match the errors of the globally 12 bit quantized versions. The difference between the L^2-errors for the horse model is due to a dilated version used by AA. Quantizations are 12 bit, based on the length of the longest edge of the object's bounding box rescaled differently for tangential and normal direction. The tan/nor column reports the optimal tangential/normal rescaling factors. FL's connectivity rates for the locally quantized models above are on average within 2% of the values of the globally quantized versions given in Table 1.*

choice of weights fits into this general framework and is the result of averaging over three parallelogram predictions over existing and virtual edges. Although it turns out to be a powerful choice in practice, we observe that different model classes prefer different weights. In particular, a first analysis shows the formation of *prediction clusters* in a space of reasonable weights - with clearly *distinct clusters for regular and irregular models*. In first experiments we found that if one has *a priory knowledge* of the class a given mesh belongs to, then compression bit rates can significantly be further reduced.

4. Conclusion and Future Work

FreeLence introduces two novel approaches for single-rate encoding of digital surfaces. On the one hand, the concept of coding with *free valences* combined with a geometry-driven traversal method yields superior bit-rates for *connectivity encoding* compared to previous approaches. As a second contribution, FreeLence introduces a simple linear *geometry prediction scheme* which yields significantly better bit rates for encoding the vertex positions, compared to existing single-rate region-growing compressors.

Experiments show that there is no universally optimal local linear prediction scheme for geometry encoding: Different model classes tend to form separate optimal prediction clusters in the moduli space of reasonable predictions. This poses two challenging problems for future work: First of all, is it possible to design optimal weights for mesh classes such as CAD models, scan models and detail-modified subdivision models? Secondly, is it possible to automate the choice of the correct weights for any such given model? The answer to these problems will depend on the search and the definition of a more *relevant pool* of 3D benchmark models enlarging the relatively small pool of examples which has commonly been used to date.

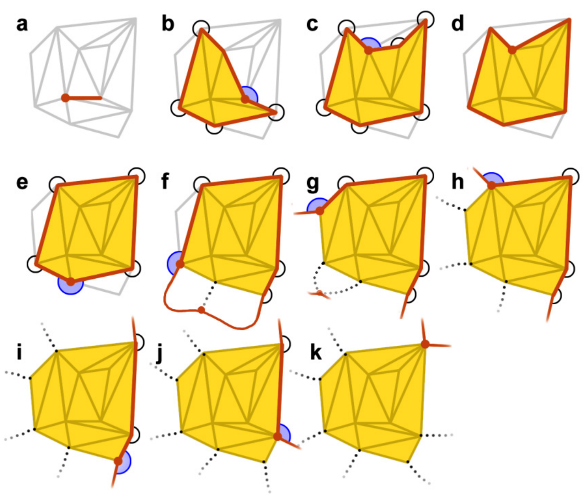

Figure 10: *A complete run of FreeLence. (**a**) Pick an initial vertex, it has valence 6, write **6**. (**b**) Close the triangles incident to the initial vertex and find a new focus according to minimal opening angle; the new focus has 2 free valences, write **2**. (**c**) Close the triangles incident to the focus and find a new focus according to minimal opening angle; the vertex previous to the focus has no free edges, write **P** and close previous. (**d**) Focus has 0 free valences, write **0**. (**e**) Focus is boundary vertex with 1 free edge preceding the boundary in clockwise order, write **D 1**. Connect a virtual edge (dashed line) to the dummy vertex. Decrease opening angles of the focus' neighbors by $\pi/2$. The total count of free valences of the focus is 1 write **1**. (**f**) New focus has 1 free valence, write **1**. (**g**) to (**j**). Focus has 0 free valences; write **0**. (**k**) Active list has length two. Stop. The full sequence is **6 2 P 0 D 1 1 1 0 0 0 0**.*

Acknowledgments. We would like to thank the anonymous reviewers for their very detailed and helpful comments, and in particular for pointing out the close relation between free valence codes and label-based methods. We thank the authors of Angle-Analyzer for providing us with their models.

References

[AD01a] ALLIEZ P., DESBRUN M.: Progressive compression for lossless transmission of triangle meshes. In *SIGGRAPH 2001 Conference Proceedings* (2001), ACM Press, pp. 195–202. 3

[AD01b] ALLIEZ P., DESBRUN M.: Valence-driven connectivity encoding for 3D meshes. In *Eurographics 2001 Conference Proceedings* (2001), vol. 20(3), pp. 480–489. 2, 3, 5

[AG03] Alliez P., Gotsman C.: Recent advances in compression of 3d meshes. In *Proceedings of the Symposium on Multiresolution in Geometric Modeling* (2003). 3

[BCG05] Ben-Chen M., Gotsman C.: On the optimality of spectral compression of mesh data. *ACM Trans. Graph. 24*, 1 (2005), 60–80. 3

[COCI02] Cohen-Or D., Cohen R., Ironi T.: *Multi-way Geometry Encoding*. Tech. rep., 2002. Technical report, School of Computer Science, Tel Aviv University. 2

[CR04] Coors V., Rossignac J.: Delphi: Geometry-based connectivity prediction in triangle mesh compression. In *The Visual Computer, International Journal of Computer Graphics* (2004), vol. 20, pp. 507Ű–520. 3

[CRS98] Cignoni P., Rocchini C., Scopigno R.: Metro: Measuring error on simplified surfaces. *Comput. Graph. Forum 17* (1998), 167–174. 7

[GA03] Gumhold S., Amjoun R.: Higher order prediction for geometry compression. In *SMI'03: Proc. Intern. Conference On Shape Modeling And Applications* (2003), pp. 59–66. 2, 7

[GD02] Gandoin P.-M., Devillers O.: Progressive lossless compression of arbitrary simplicial complexes. In *SIGGRAPH 2002 Conference Proceedings* (2002), pp. 372–379. 3

[GS98] Gumhold S., Strasser W.: Real time compression of triangle mesh connectivity. In *SIGGRAPH 1998 Conference Proceedings* (1998), ACM Press, pp. 133–140. 1, 2

[Gum99] Gumhold S.: Improved Cut-Border Machine for triangle mesh compression. In *Proceedings of Erlangen Workshop '99 on Vision, Modeling and Visualization* (1999). 3

[IA02] Isenburg M., Alliez P.: Compressing polygon mesh geometry with parallelogram prediction. In *IEEE Visualization 2002 Conference Proceedings* (2002), pp. 141–146. 2

[IGG01] Isenburg M., Gumhold S., Gotsman C.: Connectivity shapes. In *IEEE Visualization 2001 Conference Proceedings* (2001), pp. 135–142. 3

[IS99] Isenburg M., Snoeyink J.: *Spirale Reversi: Reverse decoding of the Edgebreaker encoding*. Tech. Rep. TR-99-08, 4 1999. 3

[IS00] Isenburg M., Snoeyink J.: Face fixer: Compressing polygonal meshes with properties. In *SIGGRAPH 2000 Conference Proceedings* (2000), pp. 263–270. 3

[IS04] Isenburg M., Snoeyink J.: Early-split coding of triangle mesh connectivity, 2004. submitted. 6

[Ise02] Isenburg M.: Compressing polygon mesh connectivity with degree duality prediction. In *Graphics Interface Conference Proceedings* (2002), pp. 161–170. 3

[KADS02] Khodakovsky A., Alliez P., Desbrun M., Schröder P.: Near-optimal connectivity encoding of 2-manifold polygonal meshes. *Graphics Models, special issue* (2002). 2

[KG00a] Karni Z., Gotsman C.: Spectral compression of mesh geometry. In *SIGGRAPH 2000 Conference Proceedings* (2000), ACM Press, pp. 279–286. 3

[KG00b] Kronrod B., Gotsman C.: Efficient coding of non-triangular mesh connectivity. In *PG'00: Proceedings of the 8th Pacific Conference on Computer Graphics and Applications* (2000), p. 235. 3

[KG01] Karni Z., Gotsman C.: 3D mesh compression using fixed spectral bases. In *Proc. Graphics Interface* (2001), Canadian Information Processing Society, pp. 1–8. 3

[KG02] Kronrod B., Gotsman C.: Optimized compression of triangle mesh geometry using prediction trees. In *Intern. Symp. on 3D Data Processing, Visualization and Transmission* (2002), pp. 602–608. 2, 3

[KR99] King D., Rossignac J.: Guaranteed 3.67v bit encoding of planar triangle graphs. In *Proceedings of 11th Canadian Conference on Comp. Geometry* (1999), pp. 146–149. 3

[LAD02] Lee H., Alliez P., Desbrun M.: Angle-Analyzer: A triangle-quad mesh codec. In *Eurographics conference proceedings* (2002), vol. 21, pp. 383–392. 2, 3, 4, 7, 8

[PS03] Poulalhon D., Schaeffer G.: Optimal coding and sampling of triangulations. *30th international colloquium on automata, languages and programming (ICALP'03).* (2003). 2

[Ros99] Rossignac J.: Edgebreaker: Connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics* (1999), 47–61. 1, 2

[RS99] Rossignac J., Szymczak A.: Wrap&zip decompression of the connectivity of triangle meshes compressed with Edgebreaker. *Journal of Computational Geometry 14*, 1-3 (1999), 119–135. 3

[Sha48] Shannon C. E.: A mathematical theory of communication. *The Bell System Technical J. 27* (1948). 2

[Szy02] Szymczak A.: Optimized Edgebreaker encoding for large and regular triangle meshes. In *DCC* (2002), p. 472. 3

[TG98] Touma C., Gotsman C.: Triangle mesh compression. In *Graphics Interface Conference Proceedings* (1998), pp. 26–34. 1, 2, 5, 7

[TR98] Taubin G., Rossignac J.: Just-in-time upgrades for triangle meshes. *SIGGRAPH 1998, Course Notes 21* (1998), 18–24. 2

[Tut62] Tutte W.: A census of planar triangulations. *Canadian Journal of Mathematics 14* (1962), 21–38. 2

[WNC87] Witten I. H., Neal R. M., Cleary J. G.: Arithmetic coding for data compression. *Communications of the ACM 30*, 6 (1987), 520–540. 5