Hamburger Beiträge
zur Angewandten Mathematik

Sparse Representation of Video Data by Adaptive Tetrahedralizations

Laurent Demaret, Armin Iske, Wahid Khachabi

Nr. 2009-07
May 2009

Sparse Representation of Video Data by Adaptive Tetrahedralizations

Laurent Demaret, Armin Iske, Wahid Khachabi

Abstract Natural videos are composed of a superposition of moving objects, usually resulting from anisotropic motions into different directions. By discretization with respect to time, a video may be regarded as a sequence of consecutive natural still images. Alternatively, when considering time as one dimension, a video may be viewed as a 3d scalar field. In this case, customized methods are needed for capturing both the evolution of moving contours along the time axis and the geometrical distortions of the resulting sweep surfaces. Moreover, it is desirable to work with sparse representations. Indeed, already for basic motions (e.g. rotations, translations), customized methods for the construction of well-adapted sparse video data representations are required. To this end, we propose a novel adaptive approximation algorithm for video data. The utilized nonlinear approximation scheme is based on anisotropic tetrahedralizations of the 3d video domain, whose tetrahedra are adapted locally in space (for contour-like singularities) and locally in time (for anisotropic motions). The key ingredients of our approximation method, **3AT**, are adaptive thinning, a recursive pixel removal scheme, and least squares approximation by linear splines over anisotropic tetrahedralizations. The approximation algorithm **3AT** yields a new concept for the compression of video data. We apply the proposed approximation method first to prototypical geometrical motions, before numerical simulations concerning one natural video are presented.

Laurent Demaret
HelmholtzZentrum München, Institut für Biomathematik und Biometrie (IBB), D-85764 Neuherberg, Germany, e-mail: laurent.demaret@helmholtz-muenchen.de

Armin Iske
Department of Mathematics, University of Hamburg, D-20146 Hamburg, Germany, e-mail: iske@math.uni-hamburg.de

Wahid Khachabi
Department of Mathematics, University of Hamburg, D-20146 Hamburg, Germany, e-mail: khachabi@gmail.com

1 Introduction

The production of digital video sequences is usually resulting from different technical processes involving data acquisition, projections of 3d moving objects on a camera matrix, and spatiotemporal digitalization. Natural videos typically display a superposition of anisotropic motions of different geometrical objects into different spatial directions. Simultaneous and independent motions lead to rapidly developing occlusions or disocclusions of background objects by foreground objects [1]. Therefore, the efficient representation of digital video data is a very challenging task, which in particular requires customized computational methods to cope with the very large data complexity.

When discretizing the 3d data with respect to time, a video may be regarded as a sequence of consecutive natural still images, *frames*. In the case of still images, relevant information is given by sharp contours between neighbouring objects. Consequently, in natural videos the evolution of such contours between their adjacent objects plays an important role for the human visual perception. Therefore, it is a crucial task in video processing to perform accurate representations for the moving contours.

When considering time as one (spatial) dimension, a video may also be viewed as a 3d scalar field. In this case, sweep surfaces are resulting from moving contour lines along the time axis. Despite the intrinsic three-dimensional structure of videos, commonly used video-codecs, including the popular standard method MPEG4-H264 [14], work with a still image compression method (for selected *intra-frames*) coupled with a block-oriented motion compensation (for the prediction of *inter-frames*). This strategy, however, often leads to difficulties in the following relevant situations with videos containing

- rigid motions which are not pure translations;
- object deformations (e.g. non-rigid motions by non-uniform scalings);
- object occlusions/disocclusions.

In such cases, much of the coding energy is spent on information corresponding to the compensation of the prediction error. The required information is coded by using a block-based DCT (Discrete Cosine Transform) for the inter-frames as well as for the predicted error compensation. This leads (especially at low bitrates) to strong block artefacts, partly due to discontinuities at the block boundaries, and partly due to typical Gibbs effects of the Fourier analysis method DCT. More recently, alternative wavelet-based methods were proposed to avoid strong block artefacts [15].

Alternative methods for video coding are using triangular meshes [1, 13] to handle the underlying motion field. In these more flexible coding methods, an initial triangulation is designed for the first frame. The initial triangulation is then dynamically updated according to the changes between two consecutive frames. In [1], Altunbasak and Tekalp proposed an occlusion-adaptive and content-based mesh design in combination with a forward tracking procedure. This aims at handling occlusions/disocclusions by the insertion of

new triangles or by the removal of old triangles according to the motion estimation. However, coding the corresponding motion vectors is a non-trivial problem. Moreover, in this case additional information are required to indicate covered areas (for occlusions) and uncovered areas (for disocclusions).

In this contribution, we propose a novel concept for adaptive approximation and sparse representation of video data. To this end, we regard the video as a 3d scalar field, where time is taken as one (spatial) dimension. Therefore, we essentially refrain from splitting the video data into separate consecutive image frames, unlike the above mentioned methods. We remark that our interpretation of the 3d video data does not require any sophisticated methods for explicit motion compensation, forward tracking or detection of occlusions. This gives us more flexibility in the data analysis, and moreover it reduces the computational overhead required for the maintenance of various updates between consecutive image frames.

In our approach, the video is viewed as a trivariate function, given by its discrete (greyscale) values taken at the 3d locations of the video pixels. We approximate the video by a linear spline over an adaptive tetrahedralization of the video domain. To this end, a sparse set of *significant* pixels is first adaptively chosen according to a recursive point removal scheme, adaptive thinning, such that the significant pixels capture the local motion of geometrical components of the video data. The significant pixels define a unique Delaunay tetrahedralization of the video domain, whose tetrahedra are well-adapted *locally* in space (to capture contour-like singularities) and locally in time (to capture anisotropic and irregular motions).

The Delaunay tetrahedralization yields a unique linear spline space for approximation, containing all continuous functions which are piecewise linear over the Delaunay tetrahedralization. From this spline space, we select the unique best approximation in the sense of least squares. The approximating linear spline is a *continuous* function, which can be evaluated at any point in the video domain, in particular at the *discrete* set of pixels. This allows us to reconstruct the entire video data by evaluation of the approximating linear spline at the video pixels. Note that our specific representation of the video (by a continuous function) allows us to display the reconstructed video at any subset of the (continuous) video domain.

The outline of this article is as follows. In Section 2, we discuss video data representations and Delaunay tetrahedralizations. Moreover, we show some selected prototypical motions (generated by rotations) for videos. Then, in Section 3 our video approximation scheme is explained in detail, before we discuss important computational aspects concerning its efficient implementation in Section 4. Numerical simulations are finally presented in Section 5.

2 Videos, Prototypical Motions, and Tetrahedralizations

In video sequences, variations in time are due to displacements of 3d objects projected on the video spatial domain. Many displacements can be described by elements of the Euclidean motion group. The action of Euclidean motions on the object contours produces moving surfaces. Diffusion on the 3d Euclidean motion group for the enhancement of crossing elongated structures is studied in [10].

In this section, Delaunay tetrahedralizations are introduced. Moreover, the utility of tetrahedralizations for relevant tasks of *object occlusions* and *disocclusions*, *rotations*, and *zoomings* are explained. To this end, selected prototypical motions, generated by rotations, of different objects are taken as model problems to demonstrate the enhanced flexibility of tetrahedralizations in video processing.

2.1 Representation of Video Data

To explain the representation of video data, let us first fix some notations. A greyscale video is a sequence of T rectangular planar image frames, each of size $W \times H$ pixels, so that the total number of video pixels is $N = W \times H \times T$, where each video pixel bears a greyscale (luminance) value.

Therefore, a greyscale video may be regarded as a mapping

$$V : X \longrightarrow \{0, 1, \ldots, 2^r - 1\}$$

from the 3d video domain

$$X = [0, \ldots, W-1] \times [0, \ldots, H-1] \times [0, \ldots, T-1]$$

of pixel positions to the luminance values of the greyscale video, where usually $2^r = 256$, i.e., $r = 8$. In other words, a video can be viewed as an element $V \in \{0, 1, \ldots, 2^r - 1\}^X$, where X is the set of pixels and r is the number of bits in the representation of the luminance values.

We regard a video as a trivariate function over the convex hull $[X] \subset \mathbb{R}^3$ of the pixel positions, so that $[X]$ constitutes the (continuous) parallelepipedic video domain. In this setting, each pixel in X is corresponding to a spatial grid point in $[X]$ with integer coordinates. In contrast to standard methods for video processing, our proposed approximation method is based on a pure three-dimensional interpretation of the given video data.

2.2 Delaunay Tetrahedralizations

This section introduces Delaunay tetrahedralizations, being one important ingredient of our approximation scheme. Throughout the discussion in this section, let $Y \subset \mathbb{R}^3$ denote a fixed finite point set. Recall that a tetrahedralization $\mathscr{T}_Y$ of Y is a collection of tetrahedra, whose vertex set is Y and whose union is the convex hull $[Y]$. Moreover, we assume that any pair of two distinct tetrahedra in $\mathscr{T}_Y$ intersect at most at one common vertex or along one common edge or across one common triangular face.

Delaunay tetrahedralizations are popular data structures for the efficient implementation of important 3d geometrical queries, such as nearest neighbour search or localization of closest point pairs. In this subsection, we recall some relevant properties of Delaunay tetrahedralizations.

It is convenient to introduce Delaunay tetrahedralizations through their dual *Voronoi diagrams*. To explain the duality between Voronoi diagrams and Delaunay tetrahedralizations, denote by

$$V_Y(y) = \left\{ z \in \mathbb{R}^3 \,:\, \|y - z\| = \min_{x \in Y} \|x - z\| \right\} \subset \mathbb{R}^3 \qquad \text{for } y \in Y$$

the *Voronoi tile* of $y \in Y$. Note that the Voronoi tile $V_Y(y)$ contains all points which are – w.r.t. the Euclidean norm $\|\cdot\|$ – at least as close to y as to any other point in Y. The set $\{V_Y(y)\}_{y \in Y}$ of all Voronoi tiles is called the *Voronoi diagram* of Y, yielding a partitioning of the Euclidean space, i.e.,

$$\mathbb{R}^3 = \bigcup_{y \in X} V_Y(y).$$

Note that each Voronoi tile $V_Y(y)$ is a non-empty, closed and convex polyhedron. Two different Voronoi tiles $V_Y(x)$ and $V_Y(y)$ are either disjoint or they share a vertex, an edge or a triangular face. In the latter case, the points $x \in Y$ and $y \in Y$ are said to be *Voronoi neighbours*.

By connecting all possible Voronoi neighbours, we obtain a graph whose vertex set is Y. This graph defines a *tetrahedral decomposition* $\mathscr{D}_Y$ of the convex hull $[Y]$, provided that no 5 points in X are *co-spherical*. The latter means that no 5 points in Y lie on the 2-dimensional surface of a sphere. For simplicity, we assume this property, the *Delaunay property*, until further notice.

The tetrahedral decomposition $\mathscr{D}_Y$ is said to be the *Delaunay tetrahedralization* of the point set Y. The Delaunay tetrahedralization $\mathscr{D}_X$ of X is unique. For any tetrahedron in $\mathscr{D}_Y$, its circumsphere does not contain any point from Y in its interior, according to the Delaunay property.

Finally, for any $y \in Y$, the Delaunay tetrahedralization $\mathrm{d}(Y \setminus y)$ can be computed from $\mathscr{D}_Y$ by a *local* update. This immediately follows from the Delaunay property, which implies that only the *cell* $\mathcal{C}(y)$ of y in $\mathscr{D}_Y$ needs to

be retetrahedralized. Recall that the cell $\mathcal{C}(y)$ of y is the domain consisting of all tetrahedra in $\mathscr{D}_Y$ which contain y as a vertex. For further details on Delaunay tetrahedralizations, see the textbook [17].

2.3 Sparse Representations of Prototypical Motions

This section concerns the sparse representation of prototypical motions in videos by anisotropic tetrahedralizations. For the sake of presentational simplicity and brevity, we decided to restrict ourselves to rotations of basic planar geometrical objects, ellipses and triangles, with uniform greyscale values.

We can describe the situation as follows. In the first frame, at time $t = 0$, the projection of the geometrical object is the characteristic function of a domain Ω_0 with piecewise smooth boundary Γ_0. In this particular setting, a (general) rigid motion can be viewed as a continuous mapping

$$M : [0, T] \to E^+(2)$$

from time interval $[0, T]$ to the group $E^+(2)$ of rigid motions, i.e., the group generated by translations and rotations.

Now for the special case of rotations, we assume constant angular speed, i.e., no accelerations or decelerations for further simplicity, in which case the motion

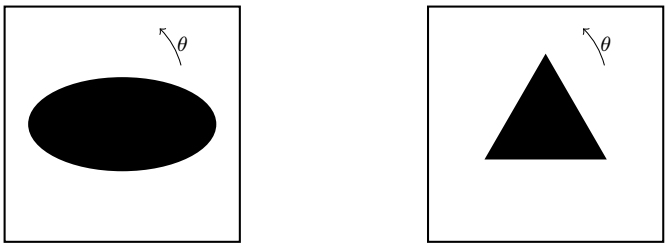
$$M(t) = M^t \in E^+(2) \qquad \text{for } t \in [0, \ldots, T]$$

can be rewritten as a planar transformation $M(u, \theta) \equiv R(\theta)$, where $R(\theta)$ is a (counter-clockwise) rotation about angle θ. In this simple case, the corresponding video sequence is fully described by the parametric surface Σ,

$$\Sigma = \bigcup_{t\in[0,T]} \Gamma_t = \bigcup_{t\in[0,T]} M(t)\Gamma_0. \tag{1}$$

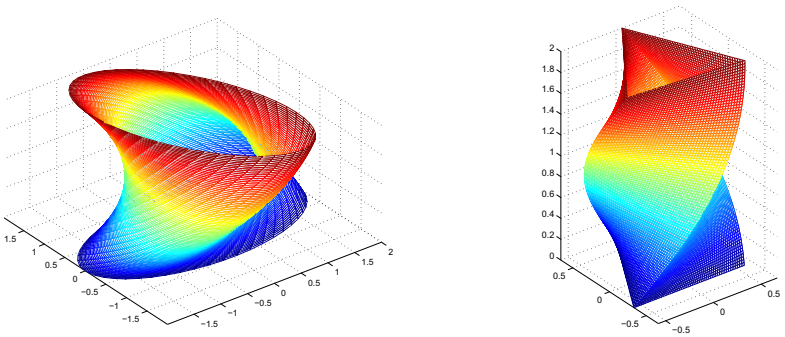
With assuming sufficiently small rotation angle θ, the surface Σ is piecewise smooth. In this case, it is reasonable to work with triangular surface elements to approximate Σ. Now the quality of the surface approximation heavily depends on the alignment of the triangular surface elements. Indeed, their long triangular edges should be aligned with directions of smaller surface curvature of Σ, whereas their short edges should point into directions with higher curvature.

Now we consider the approximation of rotational motions of two different geometrical objects, one ellipse and one equilateral triangle, see Figure 1 **(a)**,**(b)**. In either case, we let $\theta = \pi/6$ per time unit (the time between two consecutive frames). Figure 2**(a)**,**(b)** shows the resulting surfaces Σ, respectively, as defined by (1).



(a) **(b)**

Fig. 1 Two prototypical motions. **(a)** rotating ellipse; **(b)** rotating triangle. In either case, the rotation axis is aligned with the vertical (time) axis, and the origin is the center of the square domain.



(a) **(b)**

Fig. 2 Two prototypical motions. 3d visualization of the moving contours for the **(a)** rotating ellipse; **(b)** rotating triangle. In either case, the rotation axis is aligned with the vertical (time) axis, and the origin is the center of the square domain.

For the purpose of approximating the rotational triangle E, a suitable Delaunay tetrahedralization may be constructed as follows. First select a time discretization step $t_s = T/n_t$, $n_t \in \mathbb{N}$, then sample the set of vertices $Y \equiv Y(E)$ defined by

$$(E, R(t_s\theta)E, R(2t_s\theta)E, \ldots, R(n_t t_s\theta)E).$$

By their construction, the resulting $3 \times (n_t + 1)$ vertices represent the triangular shape for each of the corresponding $n_t + 1$ frames exactly. Therefore, if t_s is sufficiently small, then the corresponding Delaunay tetrahedralization of the 3d vertex set Y recovers the triangular shape throughout all frames.

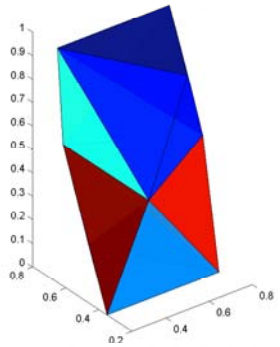


Fig. 3 Rotating triangles. Triangular surface approximation of the moving contour. Only 12 triangular faces are utilized to represent the entire video sequence, where the surface triangulation is obtained from the Delaunay tetrahedralization of 36 video pixels. In this example, we let $\theta = \pi/6$ per time step.

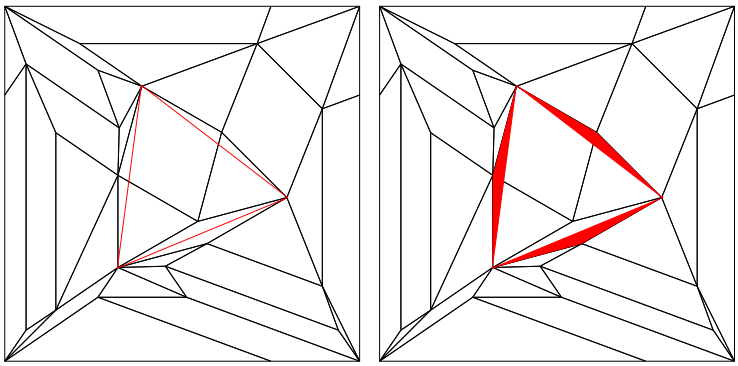


Fig. 4 Rotating triangles at time $t = 0.75$. **(a)** projection of tetrahedralization (black) and exact solution triangle (red); **(b)** the corresponding error area is filled in red.

Although this particular choice of vertices is not necessarily optimal, it leads to small approximation error of the sweep surface (moving contour).

To make one special case, let $[0, 1]$ denote the time interval, $t_s = 0.5$ and $n_t = 2$. The corresponding piecewise linear representation of the resulting sweep surface is shown in Figure 3. Moreover, Figure 4 **(a)** displays the projection of the tetrahedralization in the frame plane at intermediate time $t = 0.75$, where (for this particular example) the resulting error is maximal. The resulting error is visualized in Figure 4 **(b)**.

3 Nonlinear Approximation of Video Data

This section explains our video approximation method, **3AT**, in detail. The method **3AT** combines a recursive pixel removal scheme, adaptive thinning, with least squares approximation by trivariate linear splines. In this approximation method, adaptive thinning is first applied to obtain a sparse set of significant video pixels. The set of significant pixels yields a (unique) anisotropic Delaunay tetrahedralization, where the vertices of the tetrahedralization are given by the significant pixels. This in turn defines a linear approximation space of trivariate linear spline functions over the so obtained anisotropic tetrahedralization. The approximation to the video data is then given by the best approximating linear spline, in the sense of least squares approximation. The overall aim of the resulting adaptive approximation algorithm is to construct a suitable linear spline space of small dimension (being represented by a sparse set of significant pixels), such that the distance between the best approximating linear spline and the given video data is small, thus yielding a small reconstruction error. The construction of the spline space can be viewed as a highly nonlinear approximation process, which generalizes the algorithm proposed in [6, 7] from 2d image data to 3d video data.

In the remainder of this section, we will first discuss trivariate linear splines over tetrahedralizations, before adaptive thinning – the key ingredient of the proposed nonlinear approximation method – is introduced. This includes a discussion on the utilized significance measure, as this is required for the adaptive extraction of the significant pixels.

3.1 Linear Splines over Tetrahedralizations

Following our previous paper [7], we work with continuous and piecewise affine approximations over anisotropic tetrahedralizations. Let Π_1 denote the linear space of all trivariate polynomials of degree at most one, and let $\mathscr{D}_Y$ be a fixed Delaunay tetrahedralization. For any subset $Y \subset X$, we denote by $\mathcal{S}_Y$ the linear space of all trivariate continuous functions whose restriction to any tetrahedron $\theta \in \mathscr{D}_Y$ is affine, i.e.,

$$\mathcal{S}_Y = \{f \equiv f(x_1, x_2, x_3) \in \mathscr{C}^0([Y]) \,:\, f|_\theta \in \Pi_1 \text{ for all } \theta \in \mathscr{D}_Y\}.$$

Any element in $\mathcal{S}_Y$ is referred to as a *linear spline* over $\mathscr{D}_Y$. For given luminance values at the pixels of Y, $V\big|_Y = \{V(y) \,:\, y \in Y\}$, there is a unique linear spline interpolant $L(Y, V) \in \mathcal{S}_Y$ satisfying

$$L(Y, V)(y) = V(y) \qquad \text{for all } y \in Y.$$

The interpolant $L(Y, V)$ can be represented as a linear combination

$$L(Y,V) = \sum_{y \in Y} V(y)\varphi_y$$

of the *Courant elements* $\varphi_y \in \mathcal{S}_Y$, for $y \in Y$, being the unique Lagrangian basis functions in $\mathcal{S}_Y$ satisfying

$$\varphi_y(x) = \begin{cases} 1 & \text{for } y = x; \\ 0 & \text{for } y \neq x; \end{cases} \qquad \text{for any } x \in Y.$$

Now, for fixed $Y \subset X$ we can take the spline space $\mathcal{S}_Y$ as an approximation space for the video data $V\big|_X$, provided that the eight vertices of X (i.e., the four corners of the first frame and the four corners of the last frame) lie in Y.

3.2 Sparse Data Selection by Adaptive Thinning

We remark that the approximation quality of the video reconstruction heavily depends on the selection of the (sparse) pixel set Y. A customized construction of the sparse video data representation essentially requires an *adaptive* selection of the pixels in Y. Therefore, let us first explain how the subset $Y \subset X$ is constructed.

To obtain a suitable sparse set $Y = X_n$ of n significant pixels, for some $n \ll N$, adaptive thinning constructs a sequence of nested subsets of pixels

$$X_n \subset X_{n+1} \subset \cdots \subset X_{N-1} \subset X_N = X, \tag{2}$$

where the size $|X_p|$ of any subset X_p in (2) is p, and so $N = |X|$ is the number of pixels in X.

The utilized adaptive thinning algorithm recursively removes pixels from X, one after the other, where any removal of one pixel depends on the entire video data $V\big|_X$, as given by the luminance values attached to the video pixels. The pixel removal is done in a greedy way, where at each removal step the removed pixel is a *least significant* pixel. The generic formulation of our recursive pixel removal scheme is as follows.

Algorithm 1 (Adaptive Thinning).

(1) *Let* $X_N = X$*;*
(2) *For* $k = 1, \ldots, N - n$

(2a) *Find a least significant pixel* $x \in X_{N-k+1}$*;*
(2b) *Let* $X_{N-k} = X_{N-k+1} \setminus x$*.*

To describe a specific thinning strategy, it remains to determine a *significance measure* in order to select a *least significant* pixel in step **(2a)**. Details on this important point are discussed in the following subsection.

3.3 Significance Measures for Video Approximation

The quality of video compression schemes is measured in `dB` (decibel) by the *peak signal to noise ratio*,

$$\mathrm{PSNR} = 10 * \log_{10}\left(\frac{2^r \times 2^r}{\bar{\eta}^2(Y,X)}\right),$$

where the *mean square error* (MSE) is given by

$$\bar{\eta}^2(Y,X) = \frac{1}{|X|}\sum_{x\in X} |L(Y,V)(x) - I(x)|^2. \tag{3}$$

Therefore, to approximate the video, we wish to construct a subset $Y \subset X$, such that the resulting mean square error $\bar{\eta}^2(Y,X)$ is small. The construction of a suitable subset $Y \subset X$ is accomplished by Algorithm 1, where a natural criterion for a least significant pixel is given by the following definition, already used in our previous papers [6, 7] for pixel removal from images.

Definition 1. For $Y \subset X$, a pixel $y^* \in Y$ is said to be `least significant` in Y, iff

$$\eta(y^*) = \min_{y\in Y} \eta(y),$$

where for any $y \in Y$,

$$\eta(y) = \eta(Y \setminus y, X)$$

is the `significance` of the pixel y in Y.

In [5] we have also considered least significant pixel pairs (in images). This leads to a somewhat more sophisticated removal criterion that allows a simultaneous removal of edges (i.e., two-point removals of connected vertices). As was shown in [7], this additional option has improved the resulting image approximation quite significantly. This observation gives rise to lift the two-point removal criterion in [7] from 2d to 3d. We recall the definition of least significant pixel pairs.

Definition 2. For $Y \subset X$, a pair $\{y_1^*, y_2^*\} \subset Y$ of two pixels in Y is said to be `least significant` in Y, iff

$$\eta(y_1^*, y_2^*) = \min_{\{y_1,y_2\}\subset Y} \eta(y_1, y_2),$$

where for any pixel pair $\{y_1, y_2\} \subset Y$, we denote its `significance` in Y by

$$\eta(y_1, y_2) = \eta(Y \setminus \{y_1, y_2\}, X).$$

A pixel $y^* \in Y$ is said to be `least significant` in Y, iff it belongs to a least significant pixel pair in Y, $\{y^*, y\} \subset Y$, and satisfies $\eta(y^*) \le \eta(y)$.

As supported by our numerical comparisons in Section 5, the significance measure of Definition 2 (in comparison with that of Definition 1) improves the resulting video reconstruction considerably. Indeed, the more sophisticated significance measure of Definition 2 allows removals of edges, whose two vertices may have high individual significances (according to Definition 1), although their connecting edge may not contribute very much to the approximation to the video data (cf. [7] for more detailed explanations).

3.4 Local Optimization by Exchange

In order to further improve the quality of the significant pixels' distribution, we apply the post-processing local optimization procedure proposed in [8]. This local optimization relies on an iterative exchange of pixel pairs. At each exchange step, one current significant pixel is being swapped with one current non-significant pixel. Let us recall the definition of exchangeable pixels.

Definition 3. For any $Y \subset X$, let $Z = X \setminus Y$. A pixel pair $(y, z) \in Y \times Z$ satisfying $\eta((Y \cup z) \setminus y; X) < \eta(Y; X)$ is said to be `exchangeable`. A subset $Y \subset X$ is said to be `locally optimal` in X, iff there is no exchangeable pixel pair $(y, z) \in Y \times Z$.

By an exchange of any exchangeable pixel pair $(y, z) \in Y \times Z$, the approximation error $\eta(Y; X)$ is strictly reduced. This leads to the following local optimization algorithm which computes a locally optimal subset in X from any input $Y \subset X$.

Algorithm 2 (Exchange)
INPUT: $Y \subset X$;

(1) *Let* $Z = X \setminus Y$;
(2) **WHILE (***Y not locally optimal in X***)**

(2a) *Locate an exchangeable pair* $(y, z) \in Y \times Z$;
(2b) *Let* $Y = (Y \setminus y) \cup z$ *and* $Z = (Z \setminus z) \cup y$;

OUTPUT: $Y \subset X$, *locally optimal in* X.

In our numerical experiments, we observed that the local optimization procedure helps improve the shape of the tetrahedra, which are better adapted to the local regularity of the function. Indeed, in areas where the underlying function is smooth and convex, we obtain nearby equilateral tetrahedra, whereas in areas of smaller regularity, long and thin tetrahedra are aligned with the preference directions of the target function.

3.5 Minimization of the Mean Square Error

In a post-processing step, we further reduce the mean square error (3) by *least squares approximation* [2]. More precisely, we compute from the set $Y \subset X$ of significant pixels, output by Algorithm 1 and Algorithm 2, and from the luminance values at the pixels in X the unique *best approximation* $L^*(Y,V) \in \mathcal{S}_Y$ satisfying

$$\sum_{x\in X} |L^*(Y,V)(x) - V(x)|^2 = \min_{s\in\mathcal{S}_Y} \sum_{x\in X} |s(x) - V(x)|^2.$$

Such a best approximation exists and is unique, since $\mathcal{S}_Y$ is a finite dimensional linear space. For numerical aspects of least squares approximation we refer to the textbook [2].

4 Computational Aspects

4.1 Data Structures for Efficient Implementation

An efficient implementation of our proposed approximation method **3AT** requires suitable data structures. In the implementation of adaptive thinning, Algorithm 1, the most critical aspect concerning computational complexity is the removal of one pixel. Moreover, for the exchange of pixel pairs, by the local optimization Algorithm 2, also efficient insertions of pixels are needed. In either case, this requires efficient updates of the utilized Delaunay tetrahedralization.

Several methods have been proposed to reduce the complexity for the insertion and removal of vertices in Delaunay tetrahedralization [12, 18]. In [4], for instance, an adaptive divide-and-conquer method is suggested. Unlike the popular *Guibas-Stolfi algorithm* for two dimensions [11], the algorithm in [4] uses an incremental method in the merging step. Our implementation is based on the robust incremental *flip algorithm* of [16].

The flip algorithm in [16] performs an incremental insertion of vertices to compute the (global) Delaunay tetrahedralization. In this method, each insertion of a new vertex requires localizing a tetrahedron from the current Delaunay tetrahedralization which contains the position of that vertex. Therefore, the ordering of the points to be inserted plays an important role for the performance of the incremental insertion algorithm.

Many tetrahedralization methods merely use a random ordering of the point set. To accelerate the point insertion, a preprocessing sorting of the points is proposed in [3]. The concept in [3] relies on space-filling Hilbert curves. In our implementation vertex of removals and insertions, we follow along the lines of the ideas in [3].

Finally, we remark that any insertion or removal of a vertex from a Delaunay tetrahedralization requires only a local update. This important property of Delaunay tetrahedralizations reduces the computational comlexity of adaptive thinning, and exchange significantly. For the removal of a vertex y, for instance, only a (local) tetrahedralization of the vertex cell $C(y)$ is required (see Figure 5), where $C(y)$ is given by the union the surrounding tetrahedra of vertex y. The computational complexity of Algorithms 1,2 is discussed in Section 4.3.

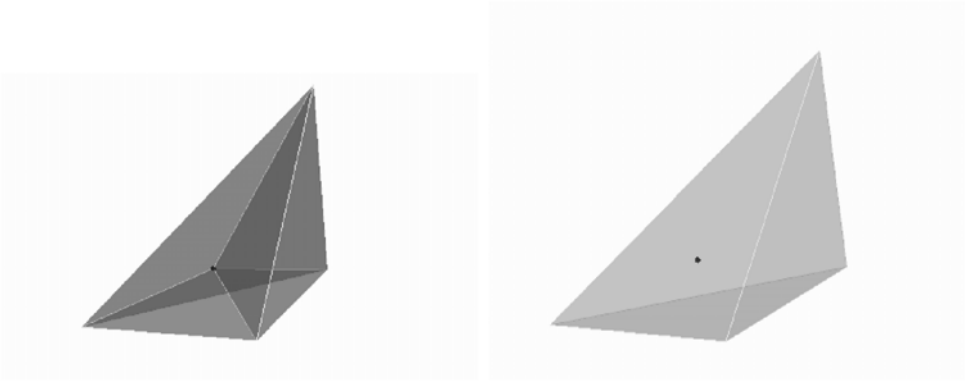


Fig. 5 Removal of a vertex from a Delaunay tetrahedralization. The update of the Delaunay tetrahedralization requires only a local retetrahedralization of the vertex cell.

4.2 Simulation of Simplicity

When five close vertices are co-spherical, the Delaunay tetrahedralization is not unique. Note that this is highly relevant in our particular situation, where the vertex positions are lying on a Cartesian grid. Moreover, uniqueness of the Delaunay tetrahedralization is an important property, which we wish to ensure for many reasons concerning the computational efficiency. In order to solve this problem, we use a generic method called *Simulation of Simplicity*, as proposed in [9]. This method serves to enforce uniqueness, especially in degenerate cases and for more general situations of co-spherical point distributions.

Unlike in other perturbation methods, the simulation of simplicity method allows us to work with integer arithmetic rather than with floating point arithmetic. We have adapted the ideas in [9] to Delaunay tetrahedralizations: by using an lexicographical order of vertices, we always obtain a unique Delaunay tetrahedralization for any given set of 3d points.

4.3 Computational Complexity

In this section, we discuss the computational complexity of our nonlinear approximation method. To this end, we show that the complexity of adaptive thinning, Algorithm 1, is $\mathcal{O}(N \log(N))$, whereas the performance of one exchange step, Algorithm 2, requires only $\mathcal{O}(\log(N))$ operations.

4.3.1 Computational Complexity of Adaptive Thinning

Let us start with the discussion on adaptive thinning. For the efficient implementation of Algorithm 1, we use two different priority queues, one for the significances of pixels and one for the significances of edges in $\mathscr{D}_Y$. Each priority queue is efficiently implemented by using the data structure *heap*.

For the significances of pixels, we use the significance measure

$$e_\delta(y) = \eta^2(y) - \eta^2(Y, X) \qquad \text{for } y \in Y.$$

We remark that the significance measure e_δ is equivalent to the pixel significance measure η of Definition 1, i.e. minimizing $\eta(y)$ among all pixels in Y is equivalent to minimizing $e_\delta(y)$ among all pixels in Y [7]. But the significance $e_\delta(y)$ is *local*, since it measures the *anticipated error*, being incurred by the removal of pixel y, on the (local) cell $C(y)$ of y.

As for the significance of pixel pairs, we work with the local significance measures

$$e_\delta(y_1, y_2) = e_\delta(y_1) + e_\delta(y_2) \qquad \text{for } [y_1, y_2] \notin \mathscr{D}_Y$$

and

$$e_\delta(y_1, y_2) = \eta^2(y_1, y_2) - \eta^2(Y, X) \qquad \text{for } [y_1, y_2] \in \mathscr{D}_Y$$

which are equivalent to the significance measure $\eta(y_1, y_2)$ in Definition 2.

Note that $e_\delta(y_1, y_2)$ is a *local* significance measure, since its computation is restricted to the union of the two cells $C(y_1)$ and $C(y_2)$. Due to the simple representation of e_δ, the maintenance of the significances $\{e_\delta(y_1, y_2) : \{y_1, y_2\} \subset Y\}$ can be reduced to the maintenance of the significances $\{e_\delta(y_1, y_2) : [y_1, y_2] \in \mathscr{D}_Y\}$ and $\{e_\delta(y) : y \in Y\}$. To this end, we employ two separate heaps.

By using the *local* significance measures e_δ (rather than η), each pixel removal (according to Algorithm 1) costs only $\mathcal{O}(1)$ operations, for the retriangulation of the cell $C(y)$ to obtain $\mathscr{D}_{Y \setminus y}$ and for the required update of the neighbouring pixels' and edges' significances. The required update for each of the two heaps costs $\mathcal{O}(\log(N))$. This makes up $\mathcal{O}(N \log(N))$ operations in total for the removal of at most N pixels.

Theorem 1. *The performance of the adaptive thinning algorithm, Algorithm 1, costs $\mathcal{O}(N \log(N))$ operations.* □

4.3.2 Computational Complexity of Exchange

Now let us turn to the complexity for one pixel exchange. In the efficient implementation of exchange we work with the *local* swapping criterion

$$e_\delta(z; Y \cup z) > e_\delta(y; Y \cup z), \quad \text{for } (y, z) \in Y \times Z. \tag{4}$$

which, for unconnected pixels in $\mathscr{D}_{Y \cup z}$, further simplifies to

$$e_\delta(z; Y \cup z) > e_\delta(y; Y), \qquad \text{for } [y; z] \notin \mathscr{D}_{Y \cup z}. \tag{5}$$

The above swapping criterion is equivalent to the criterion in Definition 3 for exchangeable pixel pairs [8].

Moreover, we work with three different heaps: one heap for the pixels in Y, `heapY`, with significances e_δ, one heap for the pixels in $Z = X \setminus Y$, `heapZ`, with significances $e_\delta(z; Y \cup z)$, for $z \in Z$, and one heap for connected pixel pairs, with significances e_δ, `heapE`. For details concerning the different significances e_δ (for pixels and edges), we refer to our previous paper [8].

The local swapping criteria (4)-(5) enables us to localize an exchangeable pixel pair in only $\mathcal{O}(1)$ operations. The subsequent update of the utilized data structures, i.e., for the three heaps `heapY`, `heapZ`, `heapE`, and the Delaunay tetrahedralization $\mathscr{D}_Y$ costs $\mathcal{O}(\log(N))$ operation. For details, we refer to [8].

Theorem 2. *The performance of one pixel exchange, Algorithm 2, costs* $\mathcal{O}(\log(N))$ *operations.* □

5 Numerical Simulations

We have applied our video approximation method **3AT** to one popular test example, called `Suzie`. The test video comprises 30 frames. The original video data is shown in Figure 7, whose 30 image frames are displayed in chronological order, from top left to bottom right.

The corresponding reconstruction of the video data by our approximation method **3AT** is shown in Figure 8. Note that **3AT** achieves to reconstruct the test data very well, especially the geometric features of the video. This is due to a well-adapted distribution of the significant pixels, as displayed in Figure 9. Their corresponding tetrahedra are shown in Figure 10, where for each frame plane only their intersecting tetrahedra are displayed. The tetrahedra are represented by their edges, where those edges lying in the frame plane are displayed by bold solid lines. The other (intersecting) edges are represented by their projections onto the corresponding frame plane, and displayed by thin solid lines.

Note that the representation of the video data by the significant pixels is very sparse. In fact, in comparison with the intermediate frames, only the first

and the last frame are containing a larger number of significant pixels, see Figure 6 (left). This is because their corresponding Delaunay triangulations (in the image frame plane) are covering the two opposite faces of the video domain, at time $t = 0$ (first frame) and at $t = 29$ (last frame). But the well-adapted distribution of the significant pixels in the intermediate frames is very efficient. Moreover, the different motions of the video's geometrical features are captured very well by the geometry of the significant pixels. The good visual quality of the video reconstruction can be evaluated through Figure 8.

Now we measure the quality of the reconstruction by the resulting PSNR value. We obtain a sparse representation of the video data by 11,430 significant pixels, (i.e., 381 significant pixels in average), yielding a PSNR value of the video reconstruction by **3AT** of 35.45 `dB`.

In addition, from the video reconstruction by **3AT**, 30 image frames were generated, each corresponding to one image frame of the given video data `Suzie`. For each of these image reconstructions, as generated by **3AT**, the corresponding number of significant pixels and the PSNR value was recorded. The two graphs in Figure 6 show the number of significant pixels (left) and the PSNR value (right), each regarded as a function of the frame indices.

As regards the evolution of significant pixels, note that the number of pixels is large between times $t = 5$ and $t = 13$. This is due to rapid motions in the video sequence, which requires more energy (in terms of number of significant pixels) to capture the relevant geometrical details.

As regards the PSNR values of the individual frames, the *minimal* PSNR value of 34.58 `dB` is attained at frame `0000`, whereas the *maximal* PSNR value of 36.32 `dB` is attained at frame `0017`. The average PSNR value is 35.49 `dB`. The resulting distribution of PSNR values (over the 30 frames) reflect a very well-balanced approximation to the video data, at remarkably good reconstruction quality and small number of significant pixels. We finally remark that our proposed approximation method features a surface-preserving smoothing process, with surfaces being understood as moving contours. This helps to avoid misaligned aliasing artefacts between consecutive frames.

Acknowledgements The second author was supported by the priority program DFG-SPP 1324 of the *Deutsche Forschungsgemeinschaft* (DFG) within the project "Adaptive Approximation Algorithms for Sparse Data Representation".

References

1. Y. Altunbasak and M. Tekalp (1997) Occlusion-adaptive content-based mesh design and forward tracking. IEEE Trans. on Image Processing **6**(9), Sep. 1997, 1270–1280.
2. Å. Björck (1996) *Numerical Methods for Least Squares Problems*. SIAM, Philadelphia.
3. J.-D. Boissonnat, O. Devillers, S. Pion, M. Teillaud, and M. Yvinec (2002) Triangulations in CGAL. Computational Geometry: Theory and Applications **22**, 5–19.
4. P. Cignoni, C. Montani, and R. Scopigno (1998) deWall: a fast divide and conquer Delaunay triangulation algorithm. Computer-Aided Design **30**(5), 333–341.
5. N. Dyn, M.S. Floater, and A. Iske (2002) Adaptive thinning for bivariate scattered data. J. Comput. Appl. Math. **145**(2), 505–517.
6. L. Demaret, N. Dyn, M. S. Floater, and A. Iske (2005) Adaptive thinning for terrain modelling and image compression. In: *Advances in Multiresolution for Geometric Modelling*, N. A. Dodgson, M. S. Floater, and M. A. Sabin (eds.), Springer, Berlin, 321–340.
7. L. Demaret, N. Dyn, and A. Iske (2006) Image compression by linear splines over adaptive triangulations. Signal Processing Journal **86**(7), July 2006, 1604–1616.
8. L. Demaret, A. Iske (2006) Adaptive image approximation by linear splines over locally optimal Delaunay triangulations. IEEE Signal Processing Letters **13**(5), May 2006, 281–284.
9. H. Edelsbrunner and E. Mücke (1990) Simulation of Simplicity: A technique to cope with degenerate cases in geometric algorithms. ACM Transactions on Graphics **9**(1), 66–104.
10. E. Franken, R. Duits, and L.M.J. Florack (2009) Diffusion on the 2d and 3d Euclidean motion group for enhancement of crossing elongated structures. This volume.
11. L. Guibas and J. Stolfi (1985) Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. ACM Transactions on Graphics **4**(2),74–123.
12. H. Ledoux, C. Gold, and G. Baciu (2005) Flipping to robustly delete a vertex in a Delaunay tetrahedralization. Computational Science and its Applications, ICCSA 2005.
13. B. Lehner, G. Umlauf, and B. Hamann (2008) Video compression using data-dependent triangulations. In: *Computer Graphics and Visualization '08*, Y. Xiao and E. ten Thij (eds.), 244–248.
14. D. Marpe, H. Schwarz, and T. Wiegand (2003) Context-based Adaptive Binary Arithmetic Coding in the H.264/AVC Video Compression Standard. IEEE Transactions on Circuits and Systems for Video Technology **13**(7), 620–636.
15. N. Mehrseresht and D. Taubman (2006) An efficient content-adaptive motion-compensated 3-D DWT with enhanced spatial and temporal scalability. IEEE Transactions on Image Processing **15**(6), 1397–1412.
16. E. Mücke (1995) A robust implementation for three-dimensional Delaunay triangulations. 1st International Computational Geometry Software Workshop, 1995.
17. F.P. Preparata and M.I. Shamos (1988) *Computational Geometry*. Springer, New York.
18. J. Shewchuk (2002) Constrained Delaunay tetrahedralizations and provably good boundary recovery. Eleventh International Meshing Roundtable (Ithaca, New York), Sandia National Laboratories, September 2002, 193–204.

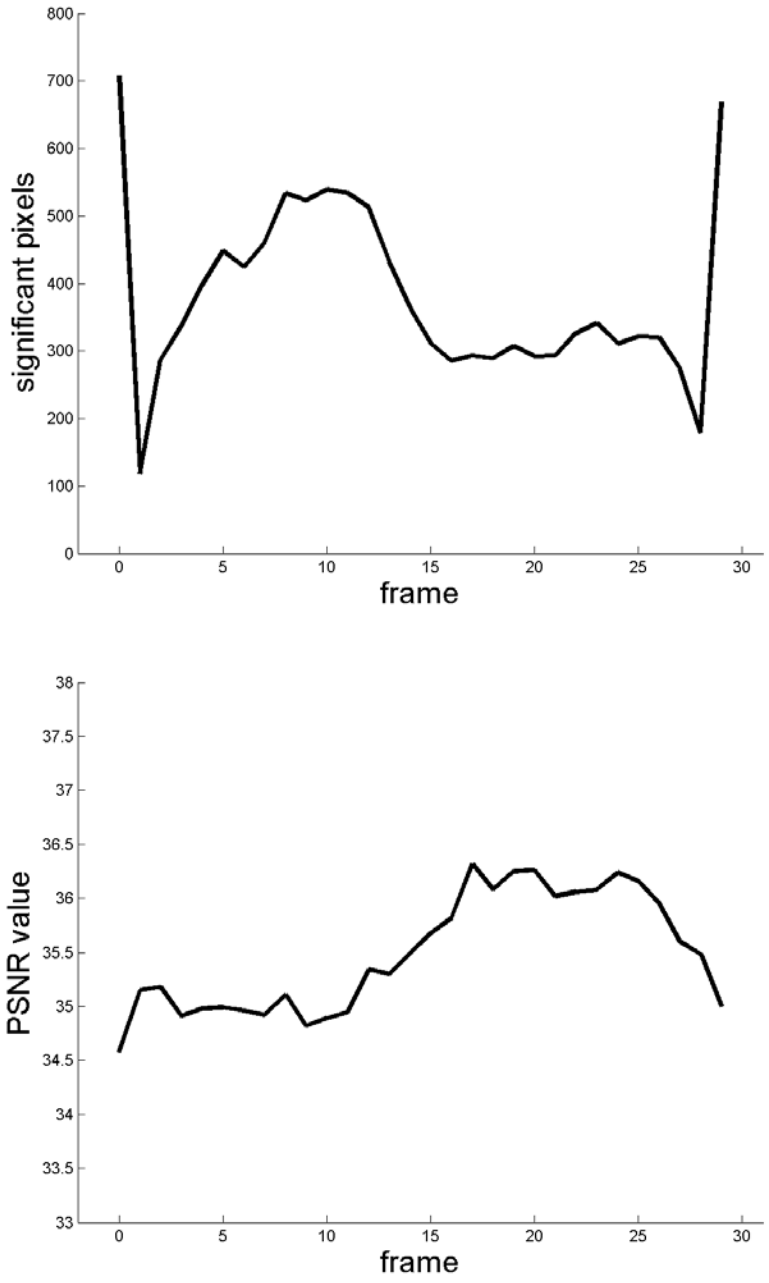


Fig. 6 `Suzie`. From the video reconstruction by **3AT**, 30 image frames were generated, each corresponding to one image frame of the given video data `Suzie`. For each of these image frames, as generated by **3AT**, the number of significant pixels and the PSNR value was recorded. The two graphs show the number of significant pixels (left); the PSNR value (right), each regarded as a function of the frame indices.

Fig. 7 `Suzie`. Video comprising 30 consecutive image frames, from top left (frame `0000`) to bottom right (frame `0029`).

Fig. 8 `Suzie`. Reconstruction of video data by our approximation method **3AT**. The reconstruction is represented by 30 consecutive image frames, displayed from top left to down right. For each image frame, the corresponding PSNR value is shown.

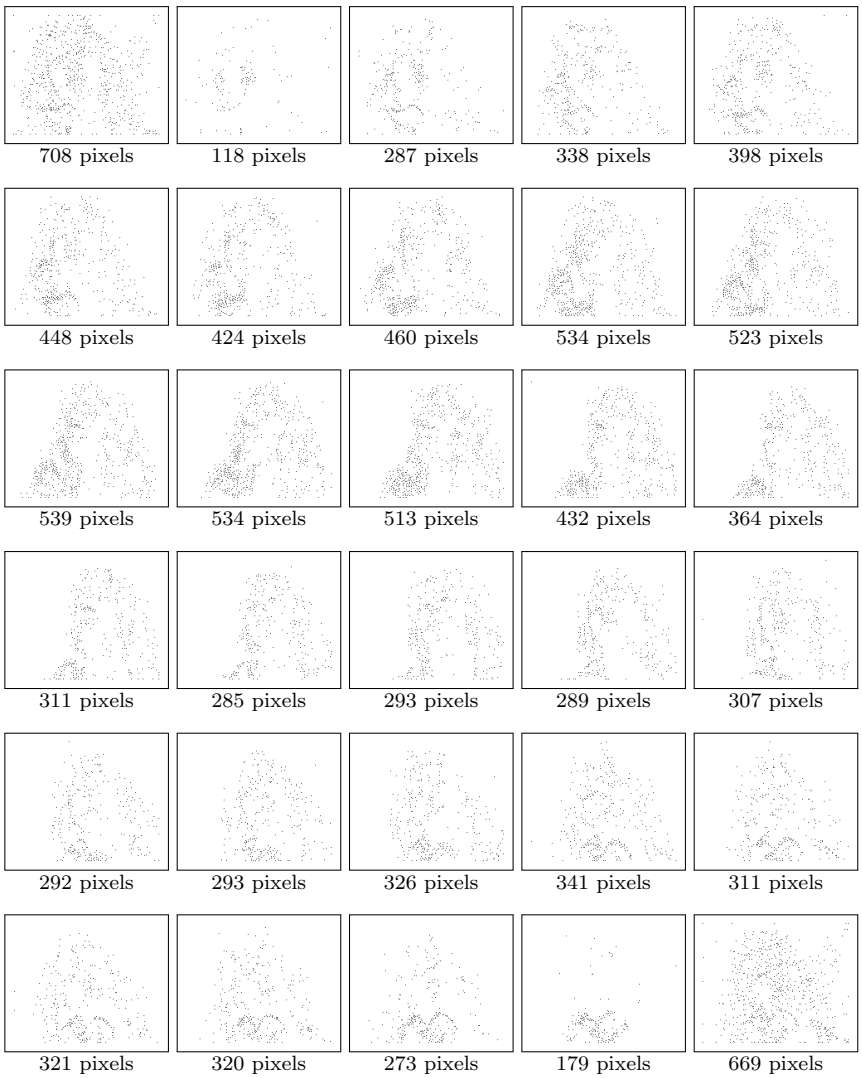


Fig. 9 `Suzie.` The significant pixels output by our approximation method **3AT**, and the number of significant pixels per frame.

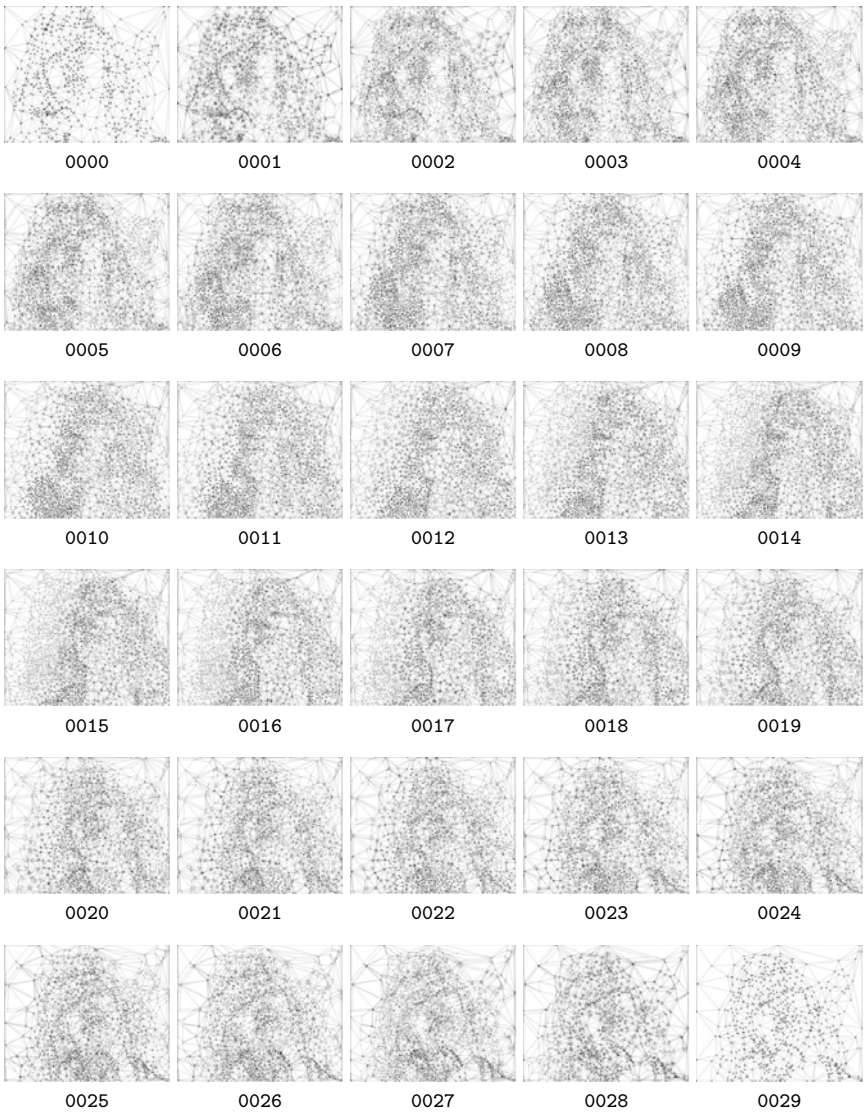


Fig. 10 `Suzie`. Tetrahedralization of the significant pixels. For each frame, only their intersecting tetrahedra are displayed. The individual tetrahedra are represented by their edges. Edges lying in the image frame are displayed by bold solid lines; edges passing through the image frame are represented by their projections onto the image plane, displayed by thin solid lines.