EFFICIENT DYNAMICAL COMPUTATION OF PRINCIPAL COMPONENTS

Darko Dimitrov[1], Mathias Holst[2], Christian Knauer[3] and Klaus Kriegel[1]

[1]*Freie Universität Berlin, Institute of Computer Science, Takustraße 9, D-14195 Berlin, Germany*

{ }

[2]*Universität Rostock, Institute of Computer Science, Albert Einstein Straße 21, D-18059 Rostock, Germany*

[3]*Universität Bayreuth, Institute of Computer Science, Universitätsstraße 30, D-95447 Bayreuth, Germany*

Keywords: Principal Component Analysis, Dynamic Computation, Bounding Box Algorithms, Approximation Algorithms.

Abstract: In this paper we consider the problem of updating principal components of a point set in $\mathbb{R}^d$ when points are added or deleted from the point set. A recent result of (Pébay, 2008) implies an efficient solution for that problem when points are added to a discrete point set. Here, we extend that result for deletions in the discrete case, and for both additions and deletions for continuous point sets in $\mathbb{R}^2$ and $\mathbb{R}^3$. In both cases, discrete and continuous, no additional data structures or storage are needed for computing the new principal components. An important application of the above results is the dynamical computation of bounding boxes based on principal component analysis. PCA bounding boxes are very often used in many fields, among others in computer graphics, for example, for ray tracing, fast rendering, collision detection, or video compression algorithms. Since some version of PCA bounding boxes have guaranties on their size (volume), they are also of interest in applications where the guaranty of the approximation quality is required. We have designed and implemented algorithms for computing dynamically PCA bounding boxes in $\mathbb{R}^3$.

1 INTRODUCTION

Principal component analysis (PCA) (Jolliffe, 2002) is probably the oldest and best known of the techniques of multivariate analysis. The central idea and motivation of PCA is to reduce the dimensionality of a point set by identifying *the most significant directions (principal components)*. Let $P = \{\vec{p}_1, \vec{p}_2, \ldots, \vec{p}_n\}$ be a set of vectors (points) in $\mathbb{R}^d$, and $\vec{\mu} = (\mu_1, \mu_2, \ldots, \mu_d) \in \mathbb{R}^d$ be the center of gravity of P. For $1 \leq k \leq d$, we use $p_{i,k}$ to denote the k-th coordinate of the vector p_i. Given two vectors $\vec{u}$ and $\vec{v}$, we use $\langle \vec{u}, \vec{v} \rangle$ to denote their inner product. For any unit vector $\vec{v} \in \mathbb{R}^d$, the *variance of P in direction $\vec{v}$* is

$$\operatorname{var}(P, \vec{v}) = \frac{1}{n} \sum_{i=1}^{n} \langle \vec{p}_i - \vec{\mu}, \vec{v} \rangle^2. \quad (1)$$

The most significant direction corresponds to the unit vector $\vec{v}_1$ such that $\operatorname{var}(P, \vec{v}_1)$ is maximum. In general, after identifying the j most significant directions $\vec{v}_1, \ldots, \vec{v}_j$, the $(j+1)$-th most significant direction corresponds to the unit vector $\vec{v}_{j+1}$ such that $\operatorname{var}(P, \vec{v}_{j+1})$ is maximum among all unit vectors perpendicular to $\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_j$.

It can be verified that for any unit vector $\vec{v} \in \mathbb{R}^d$,

$$\operatorname{var}(P, \vec{v}) = \langle \Sigma \vec{v}, \vec{v} \rangle, \quad (2)$$

where Σ is the *covariance matrix* of P. Σ is a symmetric $d \times d$ matrix where the (i, j)-th component, $\sigma_{ij}, 1 \leq i, j \leq d$, is defined as

$$\sigma_{ij} = \frac{1}{n} \sum_{k=1}^{n} (p_{k,i} - \mu_i)(p_{k,j} - \mu_j). \quad (3)$$

The procedure of finding the most significant directions, in the sense mentioned above, can be formulated as an eigenvalue problem. If $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$ are the eigenvalues of Σ, then the unit eigenvector $\vec{v}_j$ for λ_j is the j-th most significant direction. Since the matrix Σ is symmetric positive semidefinite, its eigenvectors are orthogonal, all λ_js are non-negative and $\lambda_j = \operatorname{var}(P, \vec{v}_j)$.

[1]This research was supported by the German Research Foundation (DFG), grant No. AL 253/6-2, Project "Entwurf und Analyse anwendungsbezogener geometrischer Algorithmen".

Dimitrov D., Holst M., Knauer C. and Kriegel K..
EFFICIENT DYNAMICAL COMPUTATION OF PRINCIPAL COMPONENTS.
DOI: 10.5220/0003324800850093
In *Proceedings of the International Conference on Computer Graphics Theory and Applications* (GRAPP-2011), pages 85-93
ISBN: 978-989-8425-45-4
Copyright © 2011 SCITEPRESS (Science and Technology Publications, Lda.)

Computation of the covariance matrix can be done in $O(d^2n)$ time, while computation of the eigenvalues, when d is not very large, can be done in $O(d^3)$ time, for example with the *Jacobi* or the *QR method* (Press et al., 1995). Thus, the time complexity of computing principal components of n points in $\mathbb{R}^d$ is $O(d^2n+d^3)$. The multiplicative factor of $O(d^2)$ and the additive factor of $O(d^3)$ throughout the paper will be omitted, since we will assume that d is fixed. For very large d, the problem of computing eigenvalues is non-trivial. In practice, the above mentioned methods for computing eigenvalues converge rapidly. In theory, it is unclear how to bound the running time combinatorially and how to compute the eigenvalues in decreasing order. In (Cheng and Y. Wang, 2008) a modification of the *Power method* (Parlett, 1998) is presented, which can give a guaranteed approximation of the eigenvalues with high probability.

Examples of applications of PCA include data compression, exploratory data analysis, visualization, image processing, pattern and image recognition, time series prediction, detecting perfect and reflective symmetry, and dimension detection. The thorough overview of PCA's applications can be found for example in the textbooks (Duda et al., 2001) and (Jolliffe, 2002). Most of the applications of PCA are non-geometric in their nature. However, there are also few purely geometric applications that are quite widespread in computer graphics. Example are the estimation of the undirected normals of the point sets or computing PCA bounding boxes (bounding boxes determined by the principal components of the point set).

Contributions and Organization of the Paper. Dynamic versions of the above applications, i.e., when the point set (population) changes, are of big importance and interest. Efficient solutions of those problems depend heavily on an efficient dynamic computation of the principal components (eigenvectors of the covarince matrix). Dynamic updates of variances in different settings have been studied since the sixties (Chan et al., 1979), (Knuth, 1998), (Pébay, 2008), (Welford, 1962), (West, 1979). Recently, in a technical report (Pébay, 2008) also investigated the dynamic maintenance of covariance matrices. Our contribution extends these results in the following directions:

1. We also take into account the operation of point deletions.
2. We study the dynamic computation of principal components in the continuous version.
3. We combine the dynamic PCA versions with efficient methods for computing PCA bounding boxes.

We consider the computation of the dynamic PCA bounding boxes, since it has very important applications in many fields including computer graphics, where the PCA boxes are used, for example, for ray tracing, fast rendering, video compression algorithms, or collision detection. Two distinguished hierarchical data structures from computer graphics used for representation of 3D surfaces and for rapid interference detection, based on PCA bounding boxes, are the Boxtree (Barequet et al., 1996) and the OBBTree (Gottschalk et al., 1996). We would like to stress, that PCA bounding boxes are also of interest in applications where the guaranty of the approximation quality is required, since some version of PCA bounding boxes have guaranties on their size (Dimitrov et al., 2009b). Based on the theoretical results in this paper, we have implemented several algorithms for computing PCA bounding boxes dynamically.

The organization and the main results of the paper are as follows: In Section 2 we consider the problem of updating the principal components of a set of n points, when m points are added or deleted from the point set. For both operations performed on a discrete point set in $\mathbb{R}^d$, we can compute the new principal components in $O(m)$ time for fixed d. This is a significant improvement over the commonly used approach of recomputing the principal components from scratch, which takes $O(n+m)$ time. We also consider the computation of the principal components of a dynamic continuous point set. We give closed form solutions when the point set is a convex polytope $\mathbb{R}^3$. Due to the space limitation, the cases when the point set is the boundary of a convex polytope in $\mathbb{R}^2$ or $\mathbb{R}^3$, or a convex polygon in $\mathbb{R}^2$, are left for an extended version of this paper. In Section 3 we present and verify the correctness of some theoretical results presented in the Section 2. We have implemented several dynamic PCA bounding box algorithms and evaluated their performances. Conclusion and open problems are presented in Section 4.

2 UPDATING THE PRINCIPAL COMPONENTS EFFICIENTLY

2.1 Discrete Case in $\mathbb{R}^d$

Here, we consider the problem of updating the covariance matrix Σ of a discrete point set $P = \{\vec{p}_1, \vec{p}_2, \dots, \vec{p}_n\}$ in $\mathbb{R}^d$, when m points are added or deleted from P. The recent result of (Pébay, 2008) implies the same solution that we have obtained for additions. Since the derivation of the closed form solutions for deletions is similar to that for additions, and due to the space limitation here, we just state the main result related to the discrete points sets without proof.

Theorem 2.1 *Let P be a set of n points in $\mathbb{R}^d$ with known covariance matrix Σ. Let P' be a point set in $\mathbb{R}^d$, obtained by adding or deleting m points from P. The principal components of P' can be computed in $O(m)$ time for fixed d.*

The principal components of discrete point sets can be strongly influenced by point clusters (Dimitrov et al., 2009b). To avoid the influence of the distribution of the point set, often continuous sets, especially the convex hull of a point set is considered, which lead to so-called continuous PCA. Computing PCA bounding boxes (Gottschalk et al., 1996), (Dimitrov et al., 2009a), or retrieval of 3D-objects (Vranić et al., 2001), are typical applications where continuous PCA are of interest.

2.2 Continuous Case in $\mathbb{R}^3$

Here, we consider the computation of the principal components of a dynamic continuous point set. We present a closed form-solutions when the point set is a convex polytope or the boundary of a convex polytope in $\mathbb{R}^2$ or $\mathbb{R}^3$. When the point set is the boundary of a convex polytope, we can update the new principal components in $O(k)$ time, for both deletion and addition, under the assumption that we know the k facets in which the polytope changes. Under the same assumption, when the point set is a convex polytope in $\mathbb{R}^2$ or $\mathbb{R}^3$, we can update the principal components in $O(k)$ time after adding points. But, to update the principal components after deleting points from a convex polytope in $\mathbb{R}^2$ or $\mathbb{R}^3$ we need $O(n)$ time. This is due to the fact that, after a deletion the center of gravity of the old convex hull (polyhedron) could lie outside the new convex hull, and therefore, a retetrahedralization is needed (see Subsection 2.2.1).

Due to the space limitation, we present in this section only the closed-form solutions for a convex polytope in $\mathbb{R}^3$, and leave the cases when the point set is the boundary of a convex polytope in $\mathbb{R}^2$ or $\mathbb{R}^3$, or a convex polygon in $\mathbb{R}^2$, for an extended version of this paper.

2.2.1 Continuous PCA over a Convex Polyhedron in $\mathbb{R}^3$

Let P be a point set in $\mathbb{R}^3$, and let X be its convex hull. We assume that the boundary of X is triangulated (if it is not, we can triangulate it in a preprocessing step). We choose an arbitrary point $\vec{o}$ in the interior of X, for example, we can choose $\vec{o}$ to be the center of gravity of the boundary of X. Each triangle from the boundary together with $\vec{o}$ forms a tetrahedron. Let the number of such formed tetrahedra be n. The k-th tetrahedron, with vertices $\vec{x}_{1,k}, \vec{x}_{2,k}, \vec{x}_{3,k}, \vec{x}_{4,k} = \vec{o}$, can be represented in a parametric form by $\vec{Q}_k(s,t,u) = \vec{x}_{4,k} + s\,(\vec{x}_{1,i} - \vec{x}_{4,k}) + t\,(\vec{x}_{2,i} - \vec{x}_{4,k}) + u\,(\vec{x}_{3,i} - \vec{x}_{4,k})$, for $0 \le s,t,u \le 1$, and $s+t+u \le 1$. For $1 \le i \le 3$, we use $x_{i,j,k}$ to denote the i-th coordinate of the vertex $\vec{x}_j$ of the tetrahedron $\vec{Q}_k$.

The center of gravity of the k-th tetrahedron is

$$\vec{\mu}_k = \frac{\int_0^1 \int_0^{1-s} \int_0^{1-s-t} \rho(\vec{Q}_k(s,t,u))\vec{Q}_i(s,t,u)\,du\,dt\,ds}{\int_0^1 \int_0^{1-s} \int_0^{1-s-t} \rho(\vec{Q}_k(s,t,u))\,du\,dt\,ds},$$

where $\rho(\vec{Q}_k(s,t,u))$ is a mass density at a point $\vec{Q}_k(s,t,u)$. Since we can assume $\rho(\vec{Q}_k(s,t,u)) = 1$, we have

$$\vec{\mu}_k = \frac{\int_0^1 \int_0^{1-s} \int_0^{1-s-t} \vec{Q}_k(s,t,u)\,du\,dt\,ds}{\int_0^1 \int_0^{1-s} \int_0^{1-s-t} du\,dt\,ds} = \frac{\vec{x}_{1,k}+\vec{x}_{2,k}+\vec{x}_{3,k}+\vec{x}_{4,k}}{4}.$$

The contribution of each tetrahedron to the center of gravity of X is proportional to its volume. If M_k is the 3×3 matrix whose l-th row is $\vec{x}_{l,k} - \vec{x}_{4,k}$, for $l = 1 \ldots 3$, then the volume of the k-th tetrahedron is

$$v_k = \text{volume}(Q_k) = \frac{|det(M_k)|}{3!}.$$

We introduce a weight to each tetrahedron that is proportional to its volume, define as

$$w_k = \frac{v_k}{\sum_{k=1}^n v_k} = \frac{v_k}{v},$$

where v is the volume of X. Then, the center of gravity of X is

$$\vec{\mu} = \sum_{k=1}^n w_k \vec{\mu}_k.$$

The covariance matrix of the k-th tetrahedron is

$$\begin{aligned}\Sigma_k &= \frac{\int_0^1 \int_0^{1-s} \int_0^{1-s-t} (\vec{Q}_k(s,t,u)-\vec{\mu})\,(\vec{Q}_k(s,t,u)-\vec{\mu})^T\,du\,dt\,ds}{\int_0^1 \int_0^{1-s} \int_0^{1-s-t} du\,dt\,ds} \\ &= \frac{1}{20}\Big(\textstyle\sum_{j=1}^4 \sum_{h=1}^4 (\vec{x}_{j,k} - \vec{\mu})(\vec{x}_{h,k} - \vec{\mu})^T + \\ &\qquad \textstyle\sum_{j=1}^4 (\vec{x}_{j,k} - \vec{\mu})(\vec{x}_{j,k} - \vec{\mu})^T\Big).\end{aligned}$$

The (i,j)-th element of Σ_k, $i,j \in \{1,2,3\}$, is

$$\sigma_{ij,k} = \frac{1}{20}\Big(\sum_{l=1}^4 \sum_{h=1}^4 (x_{i,l,k} - \mu_i)(x_{j,h,k} - \mu_j) + \sum_{l=1}^4 (x_{i,l,k} - \mu_i)(x_{j,l,k} - \mu_j)\Big),$$

with $\vec{\mu} = (\mu_1, \mu_2, \mu_3)$. Finally, the covariance matrix of X is

$$\Sigma = \sum_{i=1}^n w_i \Sigma_i,$$

with (i,j)-th element

$$\sigma_{ij} = \frac{1}{20}\Big(\sum_{k=1}^{n}\sum_{l=1}^{4}\sum_{h=1}^{4} w_i(x_{i,l,k}-\mu_i)(x_{j,h,k}-\mu_j)+ \sum_{k=1}^{n}\sum_{l=1}^{4} w_i(x_{i,l,k}-\mu_i)(x_{j,l,k}-\mu_j)\Big).$$

We would like to note that the above expressions hold also for any non-convex polyhedron that can be tetrahedralized. A star-shaped object, where $\vec{o}$ is the kernel of the object, is such example.

Adding Points

We add points to P, obtaining a new point set P'. Let X' be the convex hull of P'. We consider that X' is obtained from X by deleting n_d, and adding n_a tetrahedra. Let

$$\begin{aligned} v' &= \sum_{k=1}^{n} v_k + \sum_{k=n+1}^{n+n_a} v_k - \sum_{k=n+n_a+1}^{n+n_a+n_d} v_k \\ &= v + \sum_{k=n+1}^{n+n_a} v_k - \sum_{k=n+n_a+1}^{n+n_a+n_d} v_k. \end{aligned}$$

The weight of a tetrahedron Q_k is now

$$w'_k = \frac{v_k}{v'}.$$

The center of gravity of X' is

$$\begin{aligned} \vec{\mu}' &= \sum_{k=1}^{n} w'_k\vec{\mu}_k + \sum_{k=n+1}^{n+n_a} w'_k\vec{\mu}_k - \sum_{k=n+n_a+1}^{n+n_a+n_d} w'_k\vec{\mu}_k \\ &= \frac{1}{v'}\left(\sum_{k=1}^{n} v_k\vec{\mu}_k + \sum_{k=n+1}^{n+n_a} v_k\vec{\mu}_k - \sum_{k=n+n_a+1}^{n+n_a+n_d} v_k\vec{\mu}_k\right) \\ &= \frac{1}{v'}\left(v\vec{\mu} + \sum_{k=n+1}^{n+n_a} v_k\vec{\mu}_k - \sum_{k=n+n_a+1}^{n+n_a+n_d} v_k\vec{\mu}_k\right) \end{aligned} \tag{4}$$

Let

$$\vec{\mu}_a = \frac{1}{v'}\sum_{k=n+1}^{n+n_a} v_k\vec{\mu}_k, \quad \text{and} \quad \vec{\mu}_d = \frac{1}{v'}\sum_{k=n+n_a+1}^{n+n_a+n_d} v_k\vec{\mu}_k.$$

Then, we can rewrite (4) as

$$\vec{\mu}' = \frac{v}{v'}\vec{\mu} + \vec{\mu}_a - \vec{\mu}_d. \tag{5}$$

The i-th component of $\vec{\mu}_a$ and $\vec{\mu}_d$, $1 \le i \le 3$, is denoted by $\mu_{i,a}$ and $\mu_{i,d}$, respectively. The (i,j)-th component, σ'_{ij}, $1 \le i,j \le 3$, of the covariance matrix Σ' of X' is

$$\begin{aligned} \sigma'_{ij} = \frac{1}{20}\Big(&\sum_{k=1}^{n}\sum_{l=1}^{4}\sum_{h=1}^{4} w'_k(x_{i,l,k}-\mu'_i)(x_{j,h,k}-\mu'_j)+ \\ &\sum_{k=1}^{n}\sum_{l=1}^{4} w'_k(x_{i,l,k}-\mu'_i)(x_{j,l,k}-\mu'_j)+ \\ &\sum_{k=n+1}^{n+n_a}\sum_{l=1}^{4}\sum_{h=1}^{4} w'_k(x_{i,l,k}-\mu'_i)(x_{j,h,k}-\mu'_j)+ \\ &\sum_{k=n+1}^{n+n_a}\sum_{l=1}^{4} w'_k(x_{i,l,k}-\mu'_i)(x_{j,l,k}-\mu'_j)- \\ &\sum_{k=n+n_a+1}^{n+n_a+n_d}\sum_{l=1}^{4}\sum_{h=1}^{4} w'_k(x_{i,l,k}-\mu'_i)(x_{j,h,k}-\mu'_j)- \\ &\sum_{k=n+n_a+1}^{n+n_a+n_d}\sum_{l=1}^{4} w'_k(x_{i,l,k}-\mu'_i)(x_{j,l,k}-\mu'_j)\Big). \end{aligned}$$

Let

$$\sigma'_{ij} = \frac{1}{20}(\sigma'_{ij,11}+\sigma'_{ij,12}+\sigma'_{ij,21}+\sigma'_{ij,22}-\sigma'_{ij,31}-\sigma'_{ij,32}),$$

where

$$\sigma'_{ij,11} = \sum_{k=1}^{n}\sum_{l=1}^{4}\sum_{h=1}^{4} w'_k(x_{i,l,k}-\mu'_i)(x_{j,h,k}-\mu'_j), \tag{6}$$

$$\sigma'_{ij,12} = \sum_{k=1}^{n}\sum_{l=1}^{4} w'_k(x_{i,l,k}-\mu'_i)(x_{j,l,k}-\mu'_j), \tag{7}$$

$$\sigma'_{ij,21} = \sum_{k=n+1}^{n+n_a}\sum_{l=1}^{4}\sum_{h=1}^{4} w'_k(x_{i,l,k}-\mu'_i)(x_{j,h,k}-\mu'_j), \tag{8}$$

$$\sigma'_{ij,22} = \sum_{k=n+1}^{n+n_a}\sum_{l=1}^{4} w'_k(x_{i,l,k}-\mu'_i)(x_{j,l,k}-\mu'_j), \tag{9}$$

$$\sigma'_{ij,31} = \sum_{k=n+n_a+1}^{n+n_a+n_d}\sum_{l=1}^{4}\sum_{h=1}^{4} w'_k(x_{i,l,k}-\mu'_i)(x_{j,h,k}-\mu'_j), \tag{10}$$

$$\sigma'_{ij,32} = \sum_{k=n+n_a+1}^{n+n_a+n_d}\sum_{l=1}^{4} w'_k(x_{i,l,k}-\mu'_i)(x_{j,l,k}-\mu'_j). \tag{11}$$

Plugging-in the values of μ'_i and μ'_j in (6), we obtain:

$$\begin{aligned}\sigma'_{ij,11} &= \sum_{k=1}^{n}\sum_{l=1}^{4}\sum_{h=1}^{4} w'_k(x_{i,l,k} - \frac{v}{v'}\mu_i - \mu_{i,a} + \mu_{i,d})\cdot \\ &\qquad (x_{j,h,k} - \frac{v}{v'}\mu_j - \mu_{j,a} + \mu_{j,d}) \\ &= \sum_{k=1}^{n}\sum_{l=1}^{4}\sum_{h=1}^{4} w'_k(x_{i,l,k} - \mu_i + \mu_i(1-\frac{v}{v'}) - \mu_{i,a} + \mu_{i,d}) \\ &\qquad (x_{j,h,k} - \mu_j + \mu_j(1-\frac{v}{v'}) - \mu_{j,a} + \mu_{j,d}) \\ &= \sum_{k=1}^{n}\sum_{l=1}^{4}\sum_{h=1}^{4} w'_k(x_{i,l,k} - \mu_i)(x_{j,h,k} - \mu_j) \\ &+ \sum_{k=1}^{n}\sum_{l=1}^{4}\sum_{h=1}^{4} w'_k(x_{i,l,k} - \mu_i)(\mu_j(1-\frac{v}{v'}) - \mu_{j,a} + \mu_{j,d}) \\ &+ \sum_{k=1}^{n}\sum_{l=1}^{4}\sum_{h=1}^{4} w'_k(\mu_i(1-\frac{v}{v'}) - \mu_{i,a} + \mu_{i,d})(x_{j,h,k} - \mu_j) \\ &+ \sum_{k=1}^{n}\sum_{l=1}^{4}\sum_{h=1}^{4} w'_k(\mu_i(1-\frac{v}{v'}) - \mu_{i,a} + \mu_{i,d})\cdot \\ &\qquad (\mu_j(1-\frac{v}{v'}) - \mu_{j,a} + \mu_{j,d}).\end{aligned} \tag{12}$$

Since $\sum_{k=1}^{n}\sum_{l=1}^{4} w'_k(x_{i,l,k} - \mu_i) = 0$, $1 \le i \le 3$, we have

$$\begin{aligned}\sigma'_{ij,11} &= \frac{1}{v'}\sum_{k=1}^{n}\sum_{l=1}^{4}\sum_{h=1}^{4} v_k(x_{i,l,k} - \mu_i)(x_{j,h,k} - \mu_j) \\ &+ \frac{1}{v'}\sum_{k=1}^{n}\sum_{l=1}^{4}\sum_{h=1}^{4} v_k(\mu_i(1-\frac{v}{v'}) - \mu_{i,a} + \mu_{i,d})\cdot \\ &\qquad (\mu_j(1-\frac{v}{v'}) - \mu_{j,a} + \mu_{j,d}) \\ &= \frac{1}{v'}\sum_{k=1}^{n}\sum_{l=1}^{4}\sum_{h=1}^{4} v_k(x_{i,l,k} - \mu_i)(x_{j,h,k} - \mu_j) \\ &+ 16\frac{v}{v'}(\mu_i(1-\frac{v}{v'}) - \mu_{i,a} + \mu_{i,d})\cdot \\ &\qquad (\mu_j(1-\frac{v}{v'}) - \mu_{j,a} + \mu_{j,d}).\end{aligned} \tag{13}$$

Plugging-in the values of μ'_i and μ'_j in (7), we obtain:

$$\begin{aligned}\sigma'_{ij,12} &= \sum_{k=1}^{n}\sum_{l=1}^{4} w'_k(x_{i,l,k} - \frac{v}{v'}\mu_i - \mu_{i,a} + \mu_{i,d})\cdot \\ &\qquad (x_{j,h,k} - \frac{v}{v'}\mu_j - \mu_{j,a} + \mu_{j,d}) \\ &= \sum_{k=1}^{n}\sum_{l=1}^{4} w'_k(x_{i,l,k} - \mu_i + \mu_i(1-\frac{v}{v'}) - \mu_{i,a} + \mu_{i,d})\cdot \\ &\qquad (x_{j,h,k} - \mu_j + \mu_j(1-\frac{v}{v'}) - \mu_{j,a} + \mu_{j,d}) \\ &= \sum_{k=1}^{n}\sum_{l=1}^{4} w'_k(x_{i,l,k} - \mu_i)(x_{j,h,k} - \mu_j) \\ &+ \sum_{k=1}^{n}\sum_{l=1}^{4} w'_k(x_{i,l,k} - \mu_i)(\mu_j(1-\frac{v}{v'}) - \mu_{j,a} + \mu_{j,d}) \\ &+ \sum_{k=1}^{n}\sum_{l=1}^{4} w'_k(\mu_i(1-\frac{v}{v'}) - \mu_{i,a} + \mu_{i,d})(x_{j,h,k} - \mu_j) \\ &+ \sum_{k=1}^{n}\sum_{l=1}^{4} w'_k(\mu_i(1-\frac{v}{v'}) - \mu_{i,a} + \mu_{i,d})\cdot \\ &\qquad (\mu_j(1-\frac{v}{v'}) - \mu_{j,a} + \mu_{j,d}).\end{aligned} \tag{14}$$

Since $\sum_{k=1}^{n}\sum_{l=1}^{4} w'_k(x_{i,l,k} - \mu_i) = 0$, $1 \le i \le 3$, we have

$$\begin{aligned}\sigma'_{ij,12} &= \frac{1}{v'}\sum_{k=1}^{n}\sum_{l=1}^{4} v_k(x_{i,l,k} - \mu_i)(x_{j,h,k} - \mu_j) \\ &+ \frac{1}{v'}\sum_{k=1}^{n}\sum_{l=1}^{4} v_k(\mu_i(1-\frac{v}{v'}) - \mu_{i,a} + \mu_{i,d})\cdot \\ &\qquad (\mu_j(1-\frac{v}{v'}) - \mu_{j,a} + \mu_{j,d}) \\ &= \frac{1}{v'}\sum_{k=1}^{n}\sum_{l=1}^{4} v_k(x_{i,l,k} - \mu_i)(x_{j,h,k} - \mu_j) \\ &+ 4\frac{v}{v'}(\mu_i(1-\frac{v}{v'}) - \mu_{i,a} + \mu_{i,d})\cdot \\ &\qquad (\mu_j(1-\frac{v}{v'}) - \mu_{j,a} + \mu_{j,d}).\end{aligned} \tag{15}$$

From (14) and (15), we obtain

$$\begin{aligned}\sigma'_{ij,1} &= \sigma'_{ij,11} + \sigma'_{ij,12} \\ &= \sigma_{ij} + 20\frac{v}{v'}(\mu_i(1-\frac{v}{v'}) - \mu_{i,a} + \mu_{i,d})\cdot \\ &\qquad (\mu_j(1-\frac{v}{v'}) - \mu_{j,a} + \mu_{j,d}).\end{aligned} \tag{16}$$

Note that $\sigma'_{ij,1}$ can be computed in $O(1)$ time. The components $\sigma'_{ij,21}$ and $\sigma'_{ij,22}$ can be computed in $O(n_a)$ time, while $O(n_d)$ time is needed to compute $\sigma'_{ij,31}$ and $\sigma'_{ij,32}$. Thus, $\vec{\mu}'$ and

$$\begin{aligned}\sigma'_{ij} &= \frac{1}{20}(\sigma'_{ij,11} + \sigma'_{ij,12} + \sigma'_{ij,21} + \sigma'_{ij,22} + \sigma'_{ij,31} + \sigma'_{ij,32}) \\ &= \frac{1}{20}(\sigma_{ij} + \sigma'_{ij,21} + \sigma'_{ij,22} + \sigma'_{ij,31} + \sigma_{ij,32}) \\ &+ \frac{v}{v'}(\mu_i(1-\frac{v}{v'}) - \mu_{i,a} + \mu_{i,d})(\mu_j(1-\frac{v}{v'}) - \mu_{j,a} + \mu_{j,d})\end{aligned} \tag{17}$$

can be computed in $O(n_a + n_d)$ time.

Deleting Points

Let the new convex hull be obtained by deleting n_d tetrahedra from and adding n_a tetrahedra to the old convex hull. If the interior point $\vec{o}$ (needed for a tetrahedronization of a convex polytope), after several deletions, lies inside the new convex hull, then the same formulas and time complexity, as by adding points, follow. If $\vec{o}$ lie outside the new convex hull, then, we need to choose a new interior point $\vec{o}'$, and recompute the new tetrahedra associated with it. Thus, we need in total $O(n)$ time to update the principal components.

3 PRACTICAL VARIANTS OF DYNAMICAL PCA BOUNDING BOXES AND EXPERIMENTAL RESULTS

The main focus in this section is to show the advantages of the theoretical results presented in this paper in the context of computing dynamic PCA bounding boxes. We present three practical simple algorithms, and compare their performances. The algorithms were implemented in C#, C++ and OpenGL, and tested on a Core Duo 2.33GHz with 2GB memory. All algorithms use the result mentioned in Section 2.1 to compute the principal components. They differ only in how the extremal points along the principal components are found. The implemented algorithms are the following:

- **PCA-AP** (PCA-all-points) - finds the extremal points by going through all points.
- **PCA-AGP** (PCA-all-grid-points) - the space is discretized by a regular three dimensional axis-aligned grid, with cells of size $\varepsilon \times \varepsilon \times \varepsilon$. See Figure 1 for an illustration. The grid size is chosen relatively to the size of the object. Each object is scaled such that its diameter is 1. The values of ε are between 0.001 and 1. The corners of non-empty cells are candidates for extremal points along the principal directions.
- **PCA-EGP** (PCA-extremal-grid-points) - this is an improvement of the PCA-AGP algorithm. To each vertical grid line, i.e., orthogonal to the XY plane, two extremal corners of the non-empty cell are computed. Thus, we reduced the number of candidates for extremal points from $O(\frac{1}{\varepsilon^3})$ to $O(\frac{1}{\varepsilon^2})$.

We further reduce the number of points considered in the PCA-AGP and PCA-EGP algorithms by replacing the cell corners with the centers of gravity of the cells. Afterwords, we expand the resulting box by $\sqrt{3}\varepsilon/2$ to ensure that the box contains all original points. We have implemented also these variants, but, since for a reasonable big grid size ($\varepsilon \geq 0.01$) the running time improvements are negligible, we report here only the results of the base variants of the algorithms PCA-AGP and PCA-EGP. For very dense grid the improved version of the both algorithms give better results.

In the following experiments, we add (delete) random points from the point set, and compare the results of a dynamical versions of PCA bounding boxes with their corresponding static versions (when the covariance matrix of the point set is computed from scratch). The time of computing, the volume of a bounding box, and the grid density are parameters of interest in this evaluation study. The tests were performed on a large number of real graphics models taken from various publicly available sources (Stanford 3D scanning repository, 3D Cafe). Typical samples of the results are given in Table 1, Table 2, and Table 3.

The main conclusions of the experiments are as follows:

- As expected from the theoretical results, the dynamic versions of the algorithms are significantly faster than their static counterparts. Typically, the dynamic versions are about an order of magnitude faster (see Table 1).
- The dynamic PCA-AP algorithm is not only significantly faster than its static version, it is also faster than the static version of the PCA-AGP and PCA-EGP algorithms. This is due to the fact that the brute force manner of finding the extremal points is faster than computing the covariance matrix of the new point set from scratch, although both algorithms require $O(n)$ time in the asymptotic analysis.
- Clearly, the PCA-AGP and PCA-EGP algorithms, that exploit the grid subdivision structure, are faster than the PCA-AP algorithm. The price that must be paid for this is twofold. First, an extra preprocessing time for building the grid is needed. For the example considered in Table 1, computing the grid takes about 0.4 seconds for the PCA-AGP algorithm, and about 0.43 for the PCA-EGP algorithm. Second, the resulting bounding boxes are less precise (see Table 2).
- As it is shown in Table 3, for grids that are not very sparse ($\varepsilon \leq 0.03$), the approximated PCA bounding boxes computed by the PCA-AGP and PCA-EGP algorithms are quite close to the exact PCA bounding boxes.

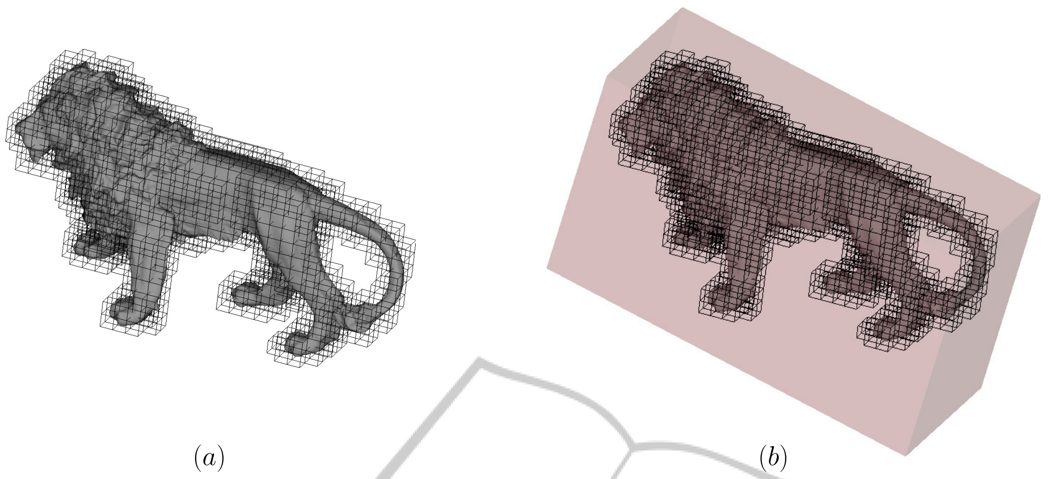


Figure 1: (a) A real world object and its corresponding grid for $\varepsilon = 0.03$. Only the non-empty cells are visualized. (b) The bounding box of the object obtained by the PCA-AGP algorithm.

Table 1: Time needed by the PCA bounding box algorithms for the lion model (183408 points). The values in the table are the average of results of 100 runs of the algorithms, each time adding/deleting the corresponding number of points.

Adding/deleting points, $\varepsilon = 0.005$						
	1pnt	1pnt	100 pnts	100 pnts	1000 pnts	1000 pnts
algorithm	static	dynamic	static	dynamic	static	dynamic
PCA-AP	0.166 s	0.014 s	0.171 s	0.015 s	0.172 s	0.016 s
PCA-AGP	0.092 s	0.010 s	0.093 s	0.009 s	0.990 s	0.017 s
PCA-EGP	0.081 s	0.006 s	0.082 s	0.006 s	0.092 s	0.014 s

Tighter bounding boxes for the PCA-AGP and PCA-EGP algorithms can be obtained by the following approach. Let P_1 be the supporting plane at the extremal grid point along one principal direction, and let P_2 be the plane parallel to P_1, such that the distance between P_1 and P_2 is $\sqrt{3}\varepsilon/2$, and P_2 intersect or is tangent to the grid. We denote by S the subspace between P_1 and P_2. Then, the candidates points for the chosen principal direction, that determine the tight bounding box, are all original points that belong to cells that have intersection with S. See Fig. 2 for an illustration. However, in the worst case all original points have to be checked.

Further (theoretical) improvement of the algorithms presented here could be obtained if, instead of the point set, we consider its convex hull when we look for extremal points. This only makes sense if the convex hull is computed dynamically. Otherwise, computing the static convex hull of the points will be more expensive than finding the exact extremal points by scanning all points.

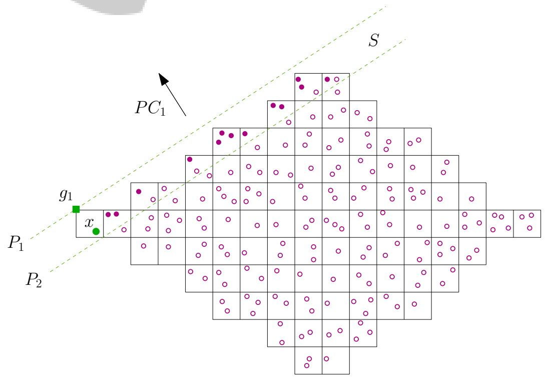


Figure 2: For the principal direction PC_1, the algorithms PCA-AGP and PCA-EGP detect the point g_1 as extremal grid point, and the point x as extremal point of the original point set. However, there are other points (the violet colored circles) that are further than x along PC_1.

3.1 Computing efficiently a Bounding Box of Several Objects

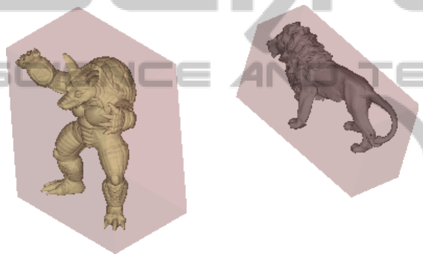
An interesting application of the closed-form solutions from Section 2 is to compute the principal components of two or more objects with already known covariance matrices. Thus, for fixed d the new covariance matrix Σ and the new principal components

Table 2: Volume of the PCA bounding box algorithms for the lion model. The values in the table are the average of results of 100 runs of the algorithms, each time adding the corresponding number of points.

Adding points, dynamic version, $\varepsilon = 0.005$					
algorithm	1pnt	10pnt	100 pnts	1000 pnts	10000 pnts
PCA-AP	285.5	644.6	856.3	1149.1	1236.4
PCA-AGP, PCA-EGP	295.5	662.7	880.3	1221.8	1263.2

Table 3: Volumes of the PCA bounding boxes algorithms for lion model for different grid density. The values in the table are the average of results of 100 runs of the algorithms, each time adding the corresponding number of points.

Adding 100 points, dynamic version						
algorithm	$\varepsilon = 0.005$	$\varepsilon = 0.01$	$\varepsilon = 0.03$	$\varepsilon = 0.05$	$\varepsilon = 0.1$	$\varepsilon = 0.2$
PCA-AP	856.3	856.3	856.3	856.3	856.3	856.3
PCA-AGP, PCA-EGP	880.3	904.3	942.3	1080.1	1292.7	2324.8


(a)

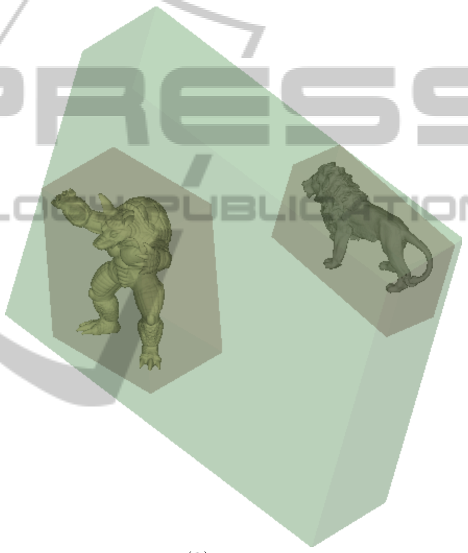

(b)

Figure 3: (a) Two objects with their PCA bounding boxes. (b) The common PCA bounding box. Computing the common PCA bounding box dynamically takes 0.004 seconds, while the static version takes 0.02 seconds.

can be computed also in $O(1)$ time.

This is a significant improvement over the commonly used approach of computing the principal components from scratch, which takes time linear in the number of points. Efficient computation of the common PCA bounding box of several objects is straightforward. See Fig. 3 for an illustration in $\mathbb{R}^3$.

4 CONCLUSIONS AND FUTURE WORK

The main contributions of this paper are the closed-form solutions for updating the principal components of a dynamic point set. The advantages of the theoretical results were verified and presented in the context of computing dynamic PCA bounding boxes, a very important application in many fields including computer graphics, where the PCA boxes are used to maintain hierarchical data structures for fast rendering of a scene or for collision detection. We have presented three practical simple algorithms and compare their performances.

An interesting open problem is to find a closed-form solution for dynamical point sets different from convex polyhedra, for example, implicit surfaces or B-splines. An implementation of computing principal components in a dynamic and continuous setting is planned for future work. Applications of the results presented here in other fields, like computer vision or visualization, are of high interest.

There are several further improvements and open problems regarding computing dynamic PCA bounding boxes. Instead of subdividing the space by a simple regular grid, one can use more sophisticated data structures, like octrees or binary space partition-trees to speed up the time needed to find the extremal points along the principal directions. A practical, implementable algorithm for computing the dynamic convex hull of the point set (computing extremal points dynamically) would also improve the dynamic PCA bounding box algorithms. Finding coresets for dynamic PCA bounding boxes will lead to efficient approximation algorithms for PCA bounding boxes. We are also not aware of data structures for efficient computation of extremal points both approximately and dynamically. Such data structures are also of interest.

ACKNOWLEDGEMENTS

The authors would like to thank the anonymous reviewers for their helpful comments and suggestions that improved the quality of the paper.

REFERENCES

Barequet, G., Chazelle, B., Guibas, L. J., Mitchell, J. S. B., and Tal, A. (1996). Boxtree: A hierarchical representation for surfaces in 3D. *Computer Graphics Forum*, 15:387–396.

Chan, T. F., Golub, G. H., and LeVeque, R. J. (1979). Updating formulae and a pairwise algorithm for computing sample variances. Technical Report STAN-CS-79-773, Department of Computer Science, Stanford University.

Cheng, S.-W. and Y. Wang, Z. W. (2008). Provable dimension detection using principal component analysis. *Int. J. Comput. Geometry Appl.*, 18:415–440.

Dimitrov, D., Holst, M., Knauer, C., and Kriegel, K. (2009a). Closed-form solutions for continuous PCA and bounding box algorithms. *A. Ranchordas et al. (Eds.): VISIGRAPP 2008, CCIS, Springer*, 24:26–40.

Dimitrov, D., Knauer, C., Kriegel, K., and Rote, G. (2009b). Bounds on the quality of the PCA bounding boxes. *Computational Geometry*, 42:772–789.

Duda, R., Hart, P., and Stork, D. (2001). *Pattern classification*. John Wiley & Sons, Inc., 2nd ed.

Gottschalk, S., Lin, M. C., and Manocha, D. (1996). OBBTree: A hierarchical structure for rapid interference detection. *Computer Graphics*, 30:171–180.

Jolliffe, I. (2002). *Principal Component Analysis*. Springer-Verlag, New York, 2nd ed.

Knuth, D. E. (1998). *The art of computer programming, volume 2: seminumerical algorithms*. Addison-Wesley, Boston, 3rd ed.

Parlett, B. N. (1998). *The symmetric eigenvalue problem*. Society of Industrial and Applied Mathematics (SIAM), Philadelphia, PA.

Pébay, P. P. (2008). Formulas for robust, one-pass parallel computation of covariances and arbitrary-order statistical moments. Technical Report SAND2008-6212, Sandia National Laboratories.

Press, W. H., Teukolsky, S. A., Veterling, W. T., and Flannery, B. P. (1995). *Numerical recipes in C: the art of scientific computing*. Cambridge University Press, New York, USA, 2nd ed.

Vranić, D. V., Saupe, D., and Richter, J. (2001). Tools for 3D-object retrieval: Karhunen-Loeve transform and spherical harmonics. In *IEEE 2001 Workshop Multimedia Signal Processing*, pages 293–298.

Welford, B. P. (1962). Note on a method for calculating corrected sums of squares and products. *Technometrics*, 4:419–420.

West, D. H. D. (1979). Updating mean and variance estimates: an improved method. *Communications of the ACM*, 22:532–535.