Scalability in Content-Based Navigation of Sound Databases

Diemo Schwarz, Norbert Schnell, Sébastien Gulluni

SCALABILITY IN CONTENT-BASED NAVIGATION OF SOUND DATABASES

Diemo Schwarz, Norbert Schnell, Sebastien Gulluni

Ircam – CNRS STMS, Paris, France

ABSTRACT

The article presents methods for sound search in large effects or instrument sound databases by interactive content-based navigation in a space of descriptors and categories, based on the principle of real-time corpus-based concatenative synthesis. We focus on three algorithms: fast similarity-based search by a *k*D-tree in the high-dimensional descriptor space, a mass–spring model with added repulsion for layout, and efficient dimensionality reduction for visualisation by hybrid multi-dimensional scaling based on these. Special attention is given to scalability to very large databases by performance evaluations and measurements. The algorithms are implemented and tested as C-libraries and Max/MSP externals within a prototype sound exploration application.

1. INTRODUCTION

Sound databases with instrumental or environmental sounds and sound effects are a vital resource for sound design, film and multi-media production, and music creation. The number and size of commercially available sound effects databases, such as *Hollywood Edge*, *Sound Ideas*, instruments or loops collections, and of community-driven on-line collections like *freesound* are growing steadily[1], with rising network bandwith, growing harddisk capacity, and falling prices for storage and distribution media accelerating this growth even further.

From a certain scale onwards, the practical problem in the exploitation of these databases is no longer the question whether a specific sound exists in the database, but how to find it. In a user survey conducted within the *Sample-Orchestrator* project[2], professional film sound designers reported about their practice of collecting the soundtracks of the *rushes*, i.e. the raw, unedited footage shot during the making of a film, to augment their collection of ambiences, reaching the mark of 1 TB of sound data, and their difficulty of finding one specific event in many long recordings with tools not adapted to such large sizes. They get by with disciplined use of manually edited metadata, and navigate by the audio waveform displayed in a sound browsing application.

Our contribution to alleviating the problems of finding the right sound in a mass of unstructured recordings is interactive navigation with immediate audio feedback in a space of sound descriptors populated by sound segments. This approach greatly speeds up the usual workflow of hierarchical menu or search mask, result list, and play/stop buttons that put many mouseclicks between the user's idea of the sound and listening to appropriate contents of the database.

In our prototype application [10], we replaced the menu- and list-driven interface with a 2D representation of a sound and category space. While navigating through the space, the sound segments close to the current position are immediately played. Playing is layered if movement is fast, so that large parts of the sound space can be explored rapidly. The strong interactivity enables the user to quickly understand the dimensions and areas of the presented space by probing sound snippets that are played as they are passed by.

This principle of navigation poses tougher requirements on the efficiency of the underlying algorithms, and on their scalability to very large databases. The article will concentrate on the two aspects of fast similarity-based search and the efficiency of low-dimensional embedding of the high-dimensional descriptor and category space[3] for visualisation, with special attention to scalability.

We chose and improved three algorithms: the *k*D-tree search algorithm (section 3), the simulation of a mass–spring–damper (MSD) system (section 4), and the hybrid multi-dimensional scaling algorithm for dimensionality reduction, that combines both of the previous (section 5). These algorithms are implemented and tested as C-libraries and Max/MSP externals as detailed in section 6.

2. RELATED WORK

The navigational approach to sound search has been inspired by interactive real-time *corpus-based concatenative synthesis* for musical creation [9, 8]. This method permits to create music by selecting snippets of a large database of pre-recorded sound by navigating through a two- or higher-dimensional space where each snippet takes up a place according to its sonic character, such as pitch, loudness, brilliance. The selected units are concatenated and played, after possibly some transformations. The method can be seen as a content-based extension to granular synthesis providing direct access to specific sound characteristics. Evidently, the

1. Since its start in 2005, http://freesound.org almost doubled every year to 62701 sounds, 681 hours, 252 GB in Februray 2009.

2. http://www.ircam.fr/306.html?&L=1

3. For our tests, we used up to 229 sound descriptors analysed by the external programs *IrcamDescriptors* and *IrcamClassifier*, partly referenced in [7], but any set of descriptors and data can be imported.

high interactivity of real-time CBCS is directly applicable to the exploration of the sounds in the corpus with the aim of searching sounds.

Related work in Music Information Retrieval start to apply graphical interfaces to content-based audio searches [1, 12] inspired by our work [8], or are concerned with the efficiency of nearest neighbour search [5, 2] or the recent method of *locality-sensitive hashing* (LSH) [11].

3. EFFICIENT NEAREST NEIGHBOUR SEARCH WITH *k*D-TREES

In any content-based retrieval application, the most recurring problem is to find the database entries most similar to a given target specification, even more so if the search is by interactive navigation through the database. In our case, the problem of finding the sound segment closest to a target point x^t in the multi-dimensional descriptor space is solved efficiently by a *branch and bound* search algorithm based on the tree-structured index provided by the *k*D-tree, representing a hierarchical decomposition of the descriptor space.

During search, whole subtrees are *pruned*, i.e. discarded from the search, by applying an elimination rule based on the farthest neighbour found so far. This removes a large amount of the distance calculations between vectors needed otherwise, resulting in a sublinear time complexity. Several variants of the algorithm are compared in [2], and it is argued that the best decomposition is along the hyperplanes orthogonal to the principal components, since it maximises the distance among the points in different subtrees and thus the probability that a subtree can be pruned.

In the following, we present the algorithms for building and searching the *k*D-tree structure, with tree node n spanning the data vector indices from p_n to q_n.

```
BUILDTREE(X = {x_0, ..., x_{N-1}})
 1  p_0 ← 0   — node 0 contains all data vectors
 2  q_0 ← N − 1
 3  for  0 ≤ l < height − 1
 4    for  2^l − 1 ≤ n < 2^{l+1} − 1
 5      DECOMPOSENODE(n, l)
 6      i ← p_n
 7      j ← q_n
 8      while  i < j   — sort node vectors
 9        while  dist_V2N(x_i, n) ≤ 0
10          i ← i + 1
11        while  dist_V2N(x_j, n) > 0
12          j ← j − 1
13        if  i < j  then
14          SWAP(i, j)
15    end
16    p_{2n+1} ← p_n   — left child of node n
17    q_{2n+1} ← j
18    p_{2n+2} ← i   — right child of node n
19    q_{2n+2} ← q_n
```

The DECOMPOSENODE(n,l) function calculates the *split plane* of node n at level l, defined by an orthogonal vector s_n and going through a point $\mu_n = \frac{1}{q_n - p_n + 1} \sum_{k=p_n}^{q_n} x_k$, that is the mean of the node's elements. This plane is used in the vector-to-node distance function $dist_{V2N}(x,n) = (x - \mu_n)/\sigma \cdot s_n$ based on the dot product. Ideally, s_n is the principal component vector of the node, but choosing it orthogonal to the axis of the dimension with the greatest variability results in an almost equally efficient search with less overhead for the decomposition.

The search algorithm uses a stack of nodes to be visited and the distance d of the target point x^t to the node's split plane in order for the elimination rule to prune child nodes when no vector closer than the current nearest neighbours can be found. It starts at node 0, which spans the whole tree, with $d = 0$.

```
SEARCHTREE(x^t, k, r)
 1  for  0 ≤ i < k  do  dist_i ← r
 2  kmax ← 0
 3  PUSH(0, 0)
 4  while  stack is not empty
 5    (d, n) ← POP()
 6    if  d ≤ dist_kmax  then
 7      if  n ≥ 2^{height−1} − 1  then  — leaf node
 8        for  p_n ≤ i ≤ q_n   — search through vectors linearly
 9          dxx ← dist_V2V(x^t, x_i)   — vector to vector distance
10          if  dxx ≤ dist_kmax  then
11            ind_kmax ← i   — indices of nearest neighbours
12            dist_kmax ← dxx
13            kmax ← argmax_{0≤j<k ∧ dist_j<r}  dist_j
14        else   — branched node
15          dxn ← dist_V2N(x^t, n)
16          if  d < 0  then
17            PUSH(2∗n+2, max(d, dxn))
18            PUSH(2∗n+1, d)
19          else
20            PUSH(2∗n+1, max(d, dxn))
21            PUSH(2∗n+2, d)
22    else  d > dist_kmax   — node can be eliminated from search
23  return  (ind, dist)
```

The radius parameter r limits the returned nearest neighbours to lie within distance r from x^t. If $r = \infty$ all k nearest neighbours are returned. Note that both distance functions dist_{V2N} and $\text{dist}_{V2V}(x,y) = (x-y)/\sigma$ can include per-descriptor-weights in σ that balance the influence of each dimension in the search, even after the tree index is built.

The performance measurements on uniformly distributed random data of size N and 10^6 random target points in figure 1 show the logarithmic time complexity of search (upper row), linear complexity for building (lower left), and the exponential influence of the number of dimensions D (lower right). However, even the worst single search time is just 2.2 ms on a 2.53 GHz Intel Core 2 processor; an initial overhead over linear search is quickly passed by with $N > 100$.

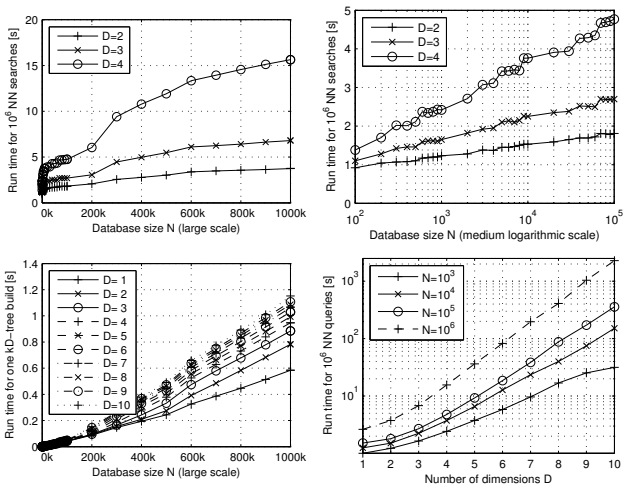

Figure 1. *k*D-tree performance measurements.

4. MASS–SPRING–DAMPER–REPULSION MODEL

Turning to the visualisation of sounds as points in a graphical interface, a useful model is the simulation of a system of masses connected by springs (or, more generally, by links). This model is the heart of the dimensionality reduction algorithm explained in section 5, but it can already be applied to an existing projection on two or three dimensions of a sound database for two purposes: First, it allows to interactively move displayed sounds with neighbouring sounds following, in order to organise the sound space. Second, the repulsion force that has been added in our implementation avoids overlapping points by pushing them apart.

The model is following MSD [3] but using an inert model for faster convergence and to avoid self-oscillating systems of masses. The physics model is given according to the implicit discretisation scheme (for the differential equations, see [3]). First, for each link between the masses at positions $x_n^{m_1}$ and $x_n^{m_2}$, the force F_{n+1} for step $n+1$ is calculated (each time step has an implicit duration of 1):

$$d_n = |x_n^{m_1} - x_n^{m_2}| \tag{1}$$

$$F_{n+1} = K\,(d_n - L) + \mu\,(d_n - d_{n-1}) + R\,\max\left(0,\, 1 - \frac{d_n}{L_R}\right) \tag{2}$$

where L is the nominal length of the link, K is the stiffness parameter depending on the *stress* $d_n - L$, and μ is the viscosity damping, depending on the change in length $d_n - d_{n-1}$. Repulsion takes place when the distance is lower than a threshold L_R and rises linearly up to R. Force F_{n+1} is along the link vector, and is added to the two masses' force vectors, negative for m_1, positive for m_2, in the direction of the link, and friction damping $\eta v_n^{m_i}$ is opposed to the velocity of mass m_i. This force is then applied to each masses position x_{n+1} and the speed v_{n+1} is updated:

$$x_{n+1} = \frac{F_{n+1}}{M} + x_n \tag{3}$$

$$v_{n+1} = x_{n+1} - x_n \tag{4}$$

The effect of the repulsion force can be seen in figure 2, where a cluster of overlapping points is distributed in space to reveal all sounds.

Figure 2. Effect of repulsion (right) on a cluster (left).

5. HYBRID MULTI-DIMENSIONAL SCALING

For the low-dimensional visualisation of a high-dimensional space, the Chalmers algorithm [4] uses a mass–spring model, where the nominal spring lengths are given by the distance in the high-dimensional data space. The basic assumption is that the final minimal stress configuration, to which the model will converge, corresponds to a good layout, where points that are close in data space are also close in layout space. An additional advantage is that the algorithm is iterative such that the successive configurations can be displayed to the user while the system converges.

Our improved hybrid algorithm consists of three phases: *1. Initialisation:* A random sample of $n_{samp} = \sqrt{N}$ points is laid out with a fully-connected mass–spring model, to provide a good starting layout for faster convergence. *2. Interpolation:* The remaining points are placed around their nearest neighbour from data space. *3. Iteration:* The placed points are laid out with a mass–spring model where n_{ngb} links are kept to the nearest neighbours in order to keep them together, and n_{rand} links are randomly chosen at each iteration, in order to move the global shape.

The choice of n_{samp} means that each iteration in the initialisation phase is linear, since a fully connected system takes $O(n_{samp}^2) = O(N)$. For interpolation, previous work [4] achieved a complexity of $O\left(N^{5/4}\right)$ by pivot-based nearest neighbour search. Here we can apply our *k*D-tree-based search, reducing the complexity to $O(N \log N)$: Our improved interpolation stage places the point at the mean position between the 3 nearest neighbours. The final iterations are sublinear, with a constant number $n_{ngb} + n_{rand}$ of links to

be evaluated, and few iterations are necessary until the total stress reaches a minimum.

6. IMPLEMENTATION

The algorithms described here are implemented as C-libraries and as externals within the FTM&CO. extension library [6] at http://ftm.ircam.fr for MAX/MSP and PUREDATA, taking advantage of FTM's real-time optimised data structures such as matrices and dictionaries. This allowed the rapid building of prototypes to test the search-by-interaction paradigm. An example sound navigation interface is shown in figure 3 and further described in [10].

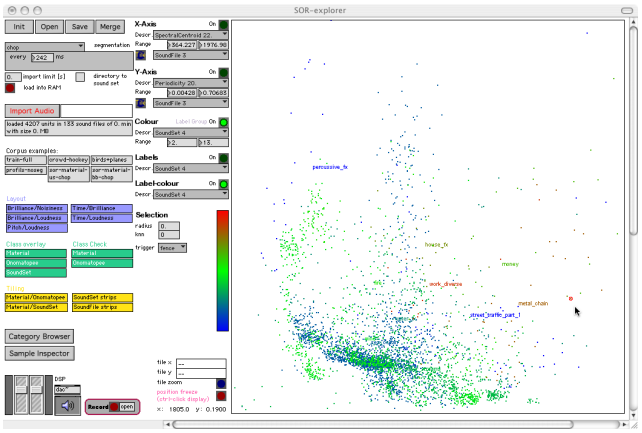

Figure 3. Screenshot of the sound navigator prototype.

7. CONCLUSION

We described three algorithms that are crucial for interactive navigation-based search in large sound databases, and their improvements, performance and implementation. First, the efficient logarithmic-time *k*D-tree search algorithm, where we added the limitation to a search radius, and weights for the descriptors. Second, the mass–spring–damper model for intuitive layout optimisation of points in a 2D interface, where we added repulsion. Third, the hybrid multi-dimensional scaling algorithm for dimensionality reduction for visualisation, is based on the MSD model, and the use of the *k*D-tree speeds up the initialisation, allows more precise pre-placement, and thus faster convergence.

All three algorithms together make the paradigm of interactive sound search by navigation scalable to very large sound databases. The prototype application contains other innovations and facilities in the user interface, such as class filters and a multi-grid visualisation, to organise search in large databases of audio descriptors and categories, that are described in an accompanying article [10].

8. ACKNOWLEDGEMENTS

The research presented here is partially funded by the French National Agency of Research ANR within the RIAM project *Sample Orchestrator*. The authors would like to thank the project partners for their fruitful collaboration, and Joel Bensoam and Bram de Jong for invaluable assistance.

9. REFERENCES

[1] G. Coleman, "Mused: Navigating the personal sample library," in *Proc. ICMC*, Copenhagen, Denmark, 2007.

[2] W. D'haes, D. van Dyck, and X. Rodet, "PCA-based branch and bound search algorithms for computing *K* nearest neighbors," *Pattern Recognition Letters*, vol. 24, no. 9–10, 2003.

[3] N. Montgermont, "Modèles physiques particulaires en environnement temps-réel : Application au contrôle des paramètres de synthèse," MSc Thesis (DEA ATIAM), University of Paris 6, 2005.

[4] A. Morrison and M. Chalmers, "Improving hybrid MDS with pivot-based searching," in *IEEE Symposium on Information Visualization*, 2003, p. 11.

[5] P. Roy *et al.*, "Exploiting the tradeoff between precision and CPU-time to speed up nearest neighbor search," in *ISMIR*, London, UK, 2005.

[6] N. Schnell, R. Borghesi, D. Schwarz, F. Bevilacqua, and R. Müller, "FTM—Complex Data Structures for Max," in *Proc. ICMC*, Barcelona, 2005.

[7] N. Schnell, A. Röbel, D. Schwarz, G. Peeters, and R. Borghesi, "MuBu & friends – assembling tools for content based real-time interactive audio processing in Max/MSP," in *Proc. ICMC*, Montreal, 2009.

[8] D. Schwarz, G. Beller, B. Verbrugghe, and S. Britton, "Real-Time Corpus-Based Concatenative Synthesis with CataRT," in *DAFx*, Montreal, 2006.

[9] D. Schwarz, "Corpus-based concatenative synthesis," *IEEE Sig. Proc. Mag.*, vol. 24, no. 2, Mar. 2007.

[10] D. Schwarz and N. Schnell, "Sound search by content-based navigation in large databases," in *Sound and Music Computing (SMC) (submitted)*, Porto, Jul. 2009.

[11] M. Slaney and M. Casey, "Locality-sensitive hashing for finding nearest neighbors," *IEEE Signal Processing Magazine*, vol. 25, no. 2, pp. 128–131, Mar. 2008.

[12] S. Streich and B. S. Ong, "A music loop explorer system," in *Proc. ICMC*, Belfast, Aug. 2008.