ISSN 1213-6972 *Volume 19, Number 1, 2011*

Journal of WSCG

An international journal of algorithms, data structures and techniques for computer graphics and visualization, surface meshing and modeling, global illumination, computer vision, image processing and pattern recognition, computational geometry, visual human interaction and virtual reality, animation, multimedia systems and applications in parallel, distributed and mobile environment.

Vaclav Skala – Union Agency

Journal of WSCG

Editor-in-Chief: Vaclav Skala
c/o University of West Bohemia, Univerzitni 8
306 14 Plzen
Czech Republic
skala@kiv.zcu.cz

Managing Editor: Vaclav Skala

Published and printed by Printed and Published by:
Vaclav Skala - Union Agency
Na Mazinach 9
CZ 322 00 Plzen
Czech Republic

Hardcopy: *ISSN 1213 – 6972*
CD ROM: *ISSN 1213 – 6980*
On-line: *ISSN 1213 – 6964*

Journal of WSCG

Editor-in-Chief

Vaclav Skala, University of West Bohemia
Centre for Computer Graphics and Visualization
Univerzitni 8, CZ 306 14 Plzen, Czech Republic
skala@kiv.zcu.cz http://herakles.zcu.cz
Journal of WSCG: URL: http://wscg.zcu.cz/jwscg

Direct Tel. +420-37-763-2473
Direct Fax. +420-37-763-2457
Fax Department: +420-37-763-2402

Editorial Advisory Board
MEMBERS

Journal of WSCG

Contents

Vol. 19, No.1

Robust Feature Point Matching in General Multi-Image Setups

Anita Sellent
TU Braunschweig, Germany
sellent@cg.tu-bs.de

Martin Eisemann
TU Braunschweig, Germany
eisemann@cg.tu-bs.de

Marcus Magnor
TU Braunschweig, Germany
magnor@cg.tu-bs.de

ABSTRACT

We present a robust feature matching approach that considers features from more than two images during matching. Traditionally, corners or feature points are matched between *pairs* of images. Starting from one image, corresponding features are searched in the other image. Yet, often this two-image matching is only a subproblem and actually robust matches over multiple views and/ or images acquired at several instants in time are required. In our feature matching approach we consider the multi-view video data modality and find matches that are consistent in three images. Requiring neither calibrated nor synchronized cameras, we are able to reduce the percentage of wrongly matched features considerably. We evaluate the approach for different feature detectors and their natural descriptors and show an application of our improved matching approach for optical flow calculation on unsynchronized stereo sequences.

Keywords: Keypoint matching, motion estimation, multi-view video.

1 INTRODUCTION

In recent years the increased availability of high quality video cameras together with readily available storage space and fast data transfer has led to a growing interest in stereoscopic or, more general, multiple view video. Although multi-view video data actually is highly redundant, many algorithms in the processing pipeline consider only pairs of images. One important processing step is establishing feature point correspondences that are used, e.g. as low-level starting point for motion estimation [SLW+10, BWSS09, BBM09]. Determination of robust feature points and corresponding feature point descriptions has been an intensely investigated area of research for decades [MTS+05, MS05]. In spite of great advances, wrongly matched correspondences are still commonly encountered. If additional information on the images is provided, e.g. by calibration, synchronization or assumption of constant rigid motion, this information can be used to eliminate wrongly matched correspondences [HZ03]. Unfortunately, in practical applications additional information is not always available as, for instance, multiple cameras are hard to synchronize in an outdoor environment and usually images of independently moving objects are recorded.

The goal of our work is to develop a versatile, robust feature point matching method that is generally applicable, e.g. also in the unconstrained multi-view video setup. Our basic idea is to exploit the redundancy in the data of multi-view video sequences with a common field of view. We use it to establish more reliable correspondences to ensure high-quality matches. Feature points are matched by considering loops of images. We introduce *three image consistent matching* and evaluate it by means of the percentage of wrong matches.

Additionally, we show how a stereo-video optical flow algorithm [SLM10] can benefit from incorporating our robustly matched features. Recent research has shown that optical flow can be improved if ideas from feature matching are included into the approach, [BBM09, XJM10]. In contrast to variational based optical flow algorithms that require an iterative approach to cope with large distances [BBPW04], features can be matched independently from their position in the image and thus deal with arbitrary distances, as long as their descriptor is sufficiently robust to the corresponding changes in perspective or object deformations. For the inclusion of feature matching, optical flow approaches pay careful attention to outier matches as these are able to prevent convergence to the desired motion fields. In this work we show that our robust loop matching strategy which exploits the data modality given for multi-view video is able to improve optical flow estimations without further outlier treatment.

2 RELATED WORK

Usually features are matched between two images from synchronized cameras and spurious matches can be discarded using epipolar geometry [SZ02, HZ03]. Generally, the assumption of global affine motion between

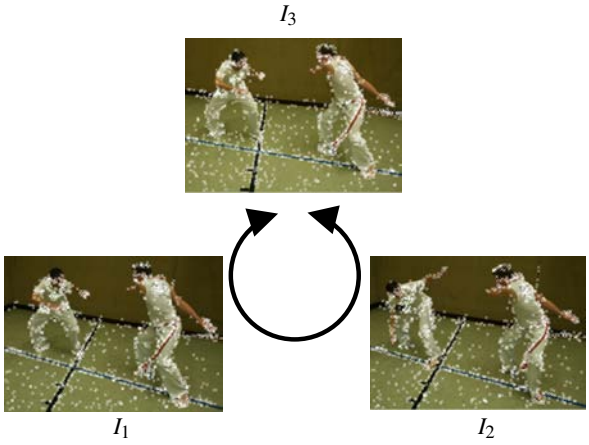


Figure 1: Three images with detected features (SIFT) of a multi-view video sequence: our algorithm accepts three images with some common field of view acquired by one or several unsynchronized and uncalibrated cameras. By requiring consistency of matches on a loop of three images, false matches are eliminated and correspondences between images can be established robustly.

two images can be used to validate matches [BGPS07]. But also game theoretic approaches exploiting local similarity transforms are used to establish reliable matchings between two images [ART10].

If several independent objects move in a monocular sequence, e.g. for person or object tracking [YJS06], feature locations from previous frames can also be used to estimate feature locations in the current frame [Zha94]. Assuming that features have at most one correct match in each frame, disjoint tracks of features over multiple frames can be considered to improve correspondences [VRB03, SS05, SSS06]. Thereby, the tracks provide a regularization of the matches over time, but no feedback for the correctness of the tracking is provided.

For static scenes, the trifocal tensor [TZ97] can be used to consider consistency of the matching between more than two images [BTZ96]. Yao and Cham first verify and add matches between image pairs to satisfy the epipolar constraint, before the matches are extended to image triples and the trifocal tensor is computed [YC07]. In contrast, Zach et al. first determine global, invertible transformations between image pairs before they detect wrong transformations on multi-image loops and discard them [ZKP10], enabling more robust multi-image static 3D reconstruction.

If a dynamic scene is recorded by multiple, unsynchronized cameras Ho and Pong work with high density feature points and use assignments of neighboring pixel in a relaxation labeling framework to obtain consistent matchings [HP96]. In the same setup, Ferrari et al. perform consistency checks on loops of images, but require an additional similarity measure that is different from the measure used to establish preliminary matchings [FTV03].

Mathematically the problem of finding consistent correspondences on three sets of equal, finite cardinality is well studied [Spi00] and approximation algorithms to the NP-hard problem have been proposed by several authors [CS92, BCS94].

In Sect. 3 we will adapt these approximation schemes to sets of different sizes. In Sect. 4 we evaluate the results of this new algorithm. We incorporate our consistent matches into a three image spatio-temporal optical flow algorithm, Sect. 5 and show how consistency of flow and features can improve dense correspondence estimation.

3 THREE IMAGE-FEATURE MATCHING

Let $I_1 : \Omega_1 \rightarrow \mathbb{R}$, $I_2 : \Omega_2 \rightarrow \mathbb{R}$ and $I_3 : \Omega_3 \rightarrow \mathbb{R}$ be three images of a multi-view video sequence that have some common field of view on a dynamic scene. In contrast to previous robust matching methods, we do not require epipolar geometry between images to be applicable, nor do we assume a temporal ordering, i.e. the three images can be acquired by one, two or three unsynchronized cameras, Fig. 1. For each image I_i, $i \in \{1,2,3\}$ a feature detector determines features $f_{i,k}, k \in \{1,\ldots,N_i\}$ with corresponding descriptors $s_{i,k}$. We denote the descriptor distance function with $d(s_{i,k}, s_{j,m})$. In our experiments, Sect. 4, we evaluate the algorithm for several detector/ descriptor variants, so we keep the description general in this section.

Usually, after detection the features are matched between two images at a time. Authors of different descriptors propose slightly different matching methods. To keep the results comparable, we follow the approach of [MS05] and use nearest neighbor matching (NN) for all two-matching steps.

A more elaborate two-matching strategy (NNDR) compares the distance of the nearest neighbor to the distance of the second nearest neighbor and only accepts a match if their ratio is below a threshold [Low04]. We also include this matching strategy into our evaluation.

If more than two images are considered, inconsistencies in the matches such as $(f_{1,k}, f_{2,m})$, $(f_{1,k}, f_{3,n})$ and $(f_{2,m}, f_{3,p})$, $p \neq n$ become obvious. In multi-view video, corresponding feature points are supposed to belong to one single scene point, so inconsistent matches indicate false matches. A straightforward approach to reduce the number of false matches is to filter out any match that is not consistent on a three image circle. To eliminate inconsistent matches already during the assignment we formulate the matching problem in a different way.

In our approach we look for *triples* $(f_{1,k}, f_{2,m}, f_{3,n})$ such that each $f_{i,j}$ is present in at most one triple. To each of the triples we assign a cost $\tilde{d}$ that is the sum of

the distances of all three descriptors $\tilde{d}(s_{1,k}, s_{2,m}, s_{3,n}) = d(s_{1,k}, s_{2,m}) + d(s_{1,k}, s_{3,n}) + d(s_{2,m}, s_{3,n})$, i.e. the distance between each pair of features is considered in the cost function, which therefore is independent of the ordering of the images. In contrast to previous approaches this formulation requires the matches in all images to be similar and thus closes the loop between the images, providing a feedback to the matching and avoiding the drift commonly encountered in considering ordered set of images. If all features were present in all three images this is an instance of the classical three-matching problem with decomposable cost-function, a NP hard problem which can be solved approximately with the following algorithm [CS92]:

i. Match the features in I_1 and I_2, e.g. using the Hungarian algorithm, (see [PS98]).

ii. Merge the sets of features on the basis of the matching in (i.) such that the new cost function between features in I_1 and I_3 is $\hat{d}(s_{1,k}, s_{3,n}) = \tilde{d}(s_{1,k}, s_{2,m}, s_{3,n})$.

iii. Match the features in I_1 and I_3 with the new distance function.

iv. Sum up all distances present in the matching.

v. Interchange the role of I_1, I_2, I_3 and restart at (i.).

vi. Of the three matchings thus obtained, return the one with the smallest sum of distances.

Note that step (ii.) enforces the third feature in the triple to be close both to the feature in I_1 and the feature in I_2. Enforcing this condition simultaneously provides the means to transport the information of the other images to the bilateral matching.

The three-match returned by this algorithm can be proved to lie within a certain distance to the actual best solution and in practice it often turns out to be the best solution [BCS94].

Yet, working with real images, we have to deal with occluded and non-detected features as well as with non-distinctive descriptors. We therefore adjust the above algorithm. In step (i.) we use NN matching or optionally NNDR matching. Additionally we match feature points only if they are mutual nearest neighbors. Thus the processing is independent from the ordering of the images and the feature points. For step (ii.) we remove all features from both images that are not matched in the previous step. We are only interested in feature points that can be matched consistently in three images. As the number of feature points differ in every image and we do not require all feature points to be matched, the sum of all matchings is no longer a reliable quality measure and step (iv.) is skipped. Correspondingly, for step (vi.) we do not return the match with the smallest overall cost, as this is dependent on the number of feature points actually matched. Instead we merge the three matches and only return those triples that are found in all three matching directions. Though this last step might seem rather restrictive, in our setup we opt for less matches with high quality instead of a higher number of matches with more questionable quality. This proceeding is in accordance with considering $\tilde{d}$ in (iii.) that enforces the matches to be mutual neighbors. In summary our algorithm looks as follows:

1. (a) Match the features in I_1 and I_2, using NN matching, optionally with distance check to the second nearest neighbor.
   (b) Match the features in I_2 and I_1, using NN matching, optionally with distance check to the second nearest neighbor.
   (c) Accept only symmetrically matched features.
2. Remove unmatched features in I_1 and merge the remaining features on the basis of the matching in (1.) such that the new cost function between matched features in I_1 and features in I_3 is $\hat{d}(s_{1,k}, s_{3,n}) = \tilde{d}(s_{1,k}, s_{2,m}, s_{3,n})$.
3. (a) Match the features in I_1 and I_3 with the new distance function using NN matching.
   (b) Match the features in I_3 and I_1 with the new distance function using NN matching.
   (c) Accept only symmetrically matched features.
4. Interchange the role of I_1, I_2, I_3 and restart at (1.).
5. Merge the three matchings and return only those matches that are assigned in all three matching directions.

4 EVALUATION OF THREE IMAGE-FEATURE MATCHING

A great number of feature detectors [MTS+05] and feature descriptors [MS05] exist in literature. For a comparison of those we refer the reader to these surveys. The aim of our work is to evaluate the impact of three-image matching and so we chose four widely used detector/ descriptor combinations for our evaluations: SIFT [Low04] and SURF [BETV08] are both scale invariant detectors for blob-like structures and with their natural descriptors also invariant to rotation and changes in illumination. We also evaluate our matching algorithm on Harris-corners [HS88] and the more recent accelerated corner detector FAST [RD06] and combine both with the normalized cross correlation (*NCC*) on a 9×9 window. We transform the normalized cross-correlation to a cost function via $d(s_{i,k}, s_{j,m}) = 1 - NCC(f_{i,k}, f_{j,m})$ to obtain a descriptor distance as used in Sect. 3. Using rather advanced and robust detectors as well as rather low level detectors we

		SIFT NN		SURF NN		Harris NN		FAST NN		SIFT NNDR	
		# M	%WM	# M	%WM	# M	%WM	# M	%WM	# M	%WM
art	2IM	1444	53.39	616	64.45	93	49.46	474	45.57	674	10.53
	3IM	603	11.28	177	20.90	44	13.64	220	13.64	506	2.57
books	2IM	1786	15.58	713	38.85	364	21.98	914	27.02	1506	2.52
	3IM	1373	2.26	318	8.81	200	9.00	517	8.70	1315	0.84
dolls	2IM	2206	23.75	809	35.60	134	18.66	812	19.33	1583	2.21
	3IM	1545	4.27	434	7.60	102	2.94	528	4.17	1367	1.02
laundry	2IM	1112	49.64	675	68.89	158	80.38	420	55.58	627	19.94
	3IM	550	15.82	193	28.50	32	40.63	174	17.24	457	7.66
moebius	2IM	1634	24.24	475	38.95	77	20.78	317	35.65	1211	4.54
	3IM	115	5.02	254	14.96	50	4.00	160	6.88	1011	2.47
reindeer	2IM	943	27.78	428	43.69	49	20.78	290	33.79	683	6.88
	3IM	664	7.08	200	14.50	37	8.11	143	11.89	578	2.77
waving	2IM	4345	11.12	1314	24.20	196	26.53	353	19.97	3804	1.26
	3IM	3995	4.76	1069	12.16	156	19.23	135	9.43	3720	0.70
stonemill	2IM	628	34.71	251	62.55	225	49.78	763	49.15	366	2.73
	3IM	427	13.11	114	35.96	133	27.82	452	22.79	324	0.62
RubberW.	2IM	2077	3.85	236	16.53	48	0.00	255	6.67	1975	0.56
	3IM	1585	0.32	107	5.61	25	0.00	153	1.31	1510	0.20
Hydr.	2IM	1111	16.56	432	20.88	176	25.57	576	22.74	853	1.52
	3IM	254	2.76	56	8.93	20	15.00	70	8.57	136	0.74
wall	2IM	7776	25.44	2365	49.26	1693	28.53	6733	33.71	5327	0.56
	3IM	5363	2.50	686	5.10	906	1.21	2892	1.87	4714	0.19
graffiti	2IM	2057	62.52	1385	77.98	265	90.68	822	91.12	689	25.83
	3IM	626	11.50	140	33.57	8	87.50	39	78.95	338	4.14

Table 1: As three image matches (3IM) have to satisfy stricter requirements than two image matches (2IM), the total number of matches is reduced while the quality of the matching is increased as the percentage of wrong matches (% WM) is considerably decreased no matter which of the feature detectors (SIFT, SURF, Harris or FAST) or matching strategy (nearest neighbor(NN) or nearest neighbor with threshold on the distance ratio (NNDR)) is used.

want to evaluate our matching scheme independently from the detector used.

For reason of comparison, in our experiments we apply nearest neighbor (NN) matching in all cases [MS05]. Additionally we apply the more advanced NNDR matching that was proposed for SIFT-features, using a threshold of 0.8 on the distance ratio [Low04]. We apply the thresholding step accordingly in the matching step (1.), but found it to have no impact in the matching step (3.) as the combined matching already is sufficiently distinguishing. We therefore do not apply the distance check in (3.).

Using a naïve MATLAB implementation on a 2.66GHz processor, three image consistent matching of 975 FAST features with 81 dimensional descriptors in I_1, 944 features in I_2 and 860 features in I_3 for the *art* scene requires 736ms. With the same setup, independent two-matchings between I_1 and I_2, I_1 and I_3 and I_2 and I_3 last together 126ms.

In our experiments we determine the number of matches and the percentage of matches outside a 5 pixel circle around the ground-truth location in different scenes. The scenes *art*, *books*, *dolls*, *laundry*, *moebius* and *reindeer* are rectified multiple view images of a static scene with known disparity [SP07]. The scenes *waving* [SLM10] and *stonemill* [LLM10] are synthetic, unsynchronized stereo sequences of a moving scene with known ground-truth correspondence fields. The scenes *RubberWhale* and *Hydrangea* are the only monocular sequences of more than two

(a) (b)

Figure 2: The two image-based matching approach (a) results in more outliers (red circles) and a lower relative amount of inliers (yellow crosses) than our three image based-matching (b). From top to bottom: scene *art* with SIFT features, *RubberWhale* with SURF features, *stonemill* with Harris corners, *laundry* with FAST features, all using nearest neighbor matching.

images with independently moving objects and known ground-truth motion from the Middlebury optical flow data set [BSL+07]. In contrast, the scenes *wall* and *graffiti* describe a viewpoint change for a static, mostly planar scene [MS05]. The number of matches and percentage of outliers are shown in Tab. 1, some examples are given in Fig. 2. As expected the number of matches is reduced with our stricter three-matching strategy. But at the same time the percentage of outliers among the assigned matches is also considerably reduced.

We also apply our algorithm to the real multi-video recordings scenes *market*, 421×452 pixel, and *capoeira*, 817×578 pixel, which are recorded using unsynchronized, uncalibrated cameras with automatic gain, while in the scene *outside*, 270×480 pixel, cameras are additionally hand-held. The algorithm is performed on the entire images with all features points found, but for visibility reasons, Fig. 3 shows the results only for 100 randomly selected SIFT-features: matched features are marked with a white x and connected via a yellow line to the location of the corresponding

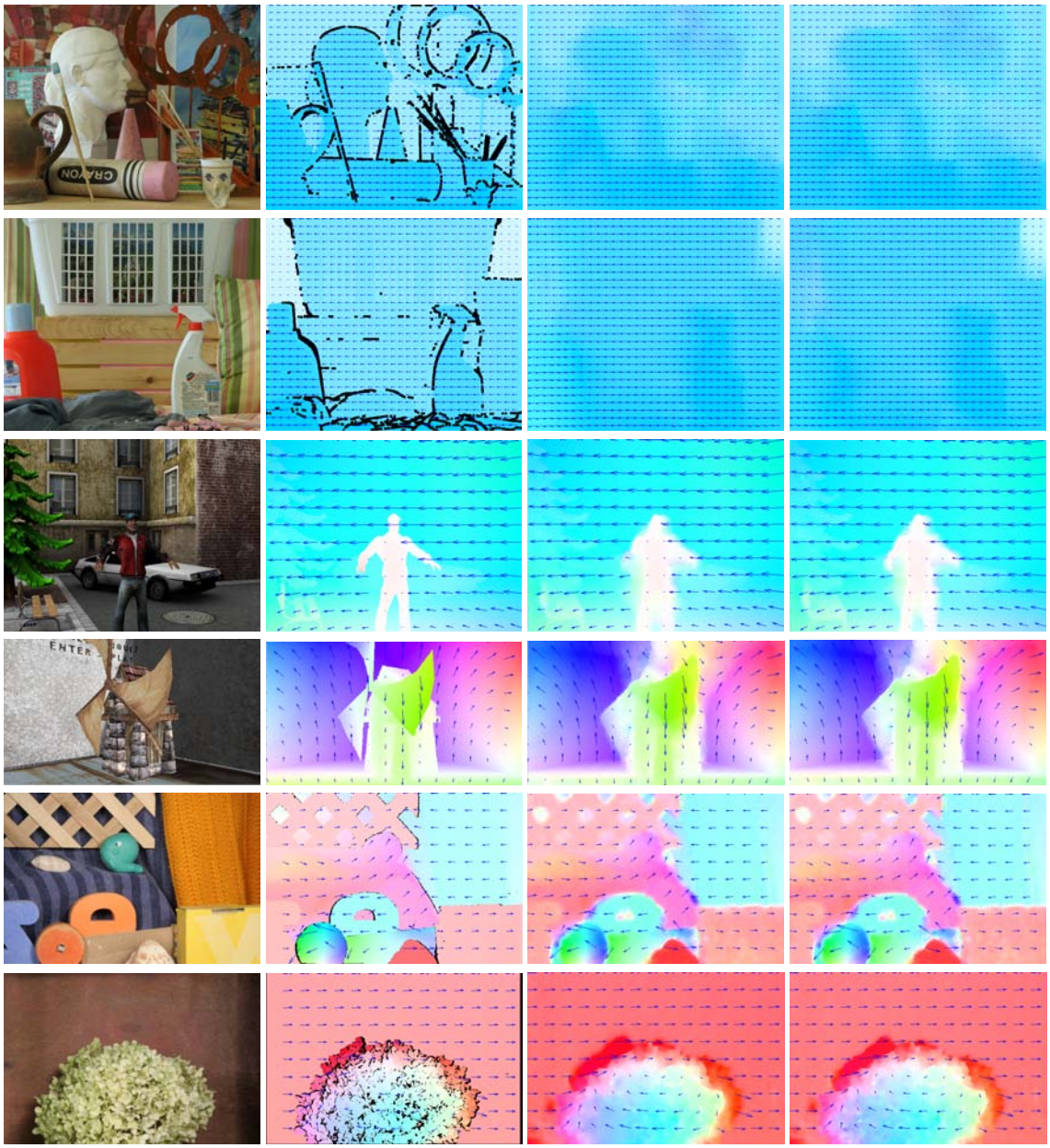
(a) (b) (c)

Figure 3: For three real world scenes *market*, *capoeira* and *outside* (a) we compare different matching strategies. Two-image matches (b) provides a larger number of matches but many outliers among them. Three-image matches (c) reduce the number of outliers considerably. For better visibility here 100 features are randomly selected and connected with the location of their matched features by a yellow line if such a feature is found.

feature. As features are only matched if they are likely correspondences in three images, the three matching algorithm obviously decreases the number of matches as compared to the algorithm that matches features based on two images. But our algorithm renounces to match many inconsistent features so that the percentage of outliers is greatly decreased. As we will show in the subsequent sections, this reduction of the relative amount of outliers allows matching based algorithms to start off much better.

5 APPLICATION TO STEREO-VIDEO CONSISTENT OPTICAL FLOW

Recent optical flow algorithms started to include feature matches into the dense correspondence estimation to faithfully detect large motion also of small objects. More specifically, Xu et al. consider motion vectors of matched features to possibly assign them to pixels all over the image [XJM10], whereas Brox et al. [BBM09] include matched regions as prior into their optical flow algorithm. We adopt the latter idea here and include matched features into the state-of-the-art optical flow for stereo sequences [SLM10]. This optical flow approach is derived from an optical flow algorithm [WTP+09] classified on the Middlebury benchmark [BSL+07]. It considers symmetry and consistency on a three image loop and therefore provides a suitable mean to evaluate the three image based matching. While in the approach of Brox et al. [WTP+09]



(a) (b) (c) (d)

Figure 4: For the scenes *art*, *laundry*, *waving*, *stonemill*, *Rubber Whale* and *Hydrangea* (a) dense ground-truth motion fields are given (b). Compared to the motion fields of the loop-consistent TV-L^2 algorithm of [SLM10], (c) the inclusion of our three-image match as prior results in motion fields with better motion detail (d).

several matches are considered to make sure that the correct correspondence is among them, we incorporate our matched features in their one-to-one fashion. Adopting the notation of $w^r_{i,j}$ for the current estimate of the motion field between image I_i and I_j we simply replace the point-wise energy E_q in [SLM10] with

$$E_f = E_q + \delta_f \|W_{i,j} - w^r_{i,j} - dw_{i,j}\|_2^2 \qquad (1)$$

where for matches $(f_{i,k}, f_{j,n}, f_{h,m})$ and $[f_{i,k}]$ the nearest integer position to the feature location

$$W_{i,j} : \Omega_i \to \mathbb{R}^2, \quad W_{i,j}(x) = \begin{cases} f_{j,n} - f_{i,k} & \text{if } x = [f_{i,k}] \\ 0 & \text{else} \end{cases} \qquad (2)$$

is a function that describes the matching of the features, $\mu, c > 0$ constants and

$$\delta_f(x) = \mu \begin{cases} 1 - \arctan \frac{\tilde{d}(s_{i,k}, s_{j,n})}{c2\pi} & \text{if } x = [f_{i,k}] \\ 0 & \text{else} \end{cases} \qquad (3)$$

a function that assigns values depending on the matching costs or 0 to each point in Ω_i. This new energy is still a quadratic function in the update $dw_{i,j}$, so the updating scheme of [SLM10] is maintained. Note that for all experiments we fix $\mu = 10^3$ and $c = \frac{1}{5}$

To speed up calculations and assist the determination of large flows, loop consistent flow estimation is performed on a factor 0.5 image pyramid. Similar to [BBM09] we down-sample the prior $W_{i,j}$ by considering the 2×2 pixels that are represented by one single pixel in the next coarser level. From the four pixels in the finer level we only pass on to the next coarser level half the motion and the weight of the pixel with the highest weight $\delta_f(x)$. Thus, if no other matches are found in the vicinity, the original match is propagated to the next coarser level or else the match with the smallest cost is used. Having thus established a matching-based prior on all levels of a scale pyramid, we initialize the dense flows on the coarsest level with zero and perform 10 iterations of the updating scheme before proceeding to the next finer level. We use the upscaled flow field from the previous level as initialization on the finer level and thus proceed till the original resolution is reached.

5.1 Evaluation

To evaluate the impact of three image-consistent matching on optical flow estimation, we use all the data sets with known ground-truth motion from Sect. 4 except for the scenes *graffiti* and *wall* which only contain camera motion around a planar scene and are therefore of no interest for dense motion field estimation. We measure the average angular error (AAE) and average endpoint error (AEE) [BSL$^+$07] between the computed and the ground-truth displacement fields. For comparison, we also calculate flow fields with a two-image TV-L^2 approach [SLM10] incorporating standard two image-feature matching as prior and the three image-loop consistent optical flow algorithm [SLM10] without prior. As SURF features provide the best cover of our test scenes with feature points, we here only show the results obtained with SURF. Flow fields incorporating priors obtained with other descriptors behave qualitatively in the same way:

If only two image matches and forward flow are considered, wrong matches have a strong impact and lead to results with high error, Tab. 2. In [SLM10] Sellent et al. show that loop consistent flow improves the results of the TV-L^2 approach. Incorporating feature points that are likewise consistent on three images is able to further improve the results. An improvement is also visible in the flow field, Fig. 4, as small structures such as e.g. the hand in the *waving* scene are better preserved than without the prior matches.

6 CONCLUSIONS AND FUTURE WORK

In our article we show that even in the absence of camera calibration and synchronization, feature points can be matched more robustly if three images are considered simultaneously. By requiring that features are consistent in three images, the quality of the matching improves as the percentage of wrong matches is considerably reduced.

We also combine three-image matching with three image-loop consistent optical flow estimation and obtain dense flow fields that have a smaller error and better preserved motion details than either the loop-consistent flow or basic flow with non-robustly matched features.

In this work we extend the traditional two image approach to three images and obtain more robust results. Future work in this direction compromises to evaluate whether this trend can be continued if four or more images are used and whether there is an optimal number of images to be used.

ACKNOWLEDGEMENTS

This work has been funded by the German Science Foundation, DFG MA2555/4-2.

REFERENCES

[ART10] A. Albarelli, E. Rodolà, and A. Torsello. Robust game-theoretic inlier selection for bundle adjustment. In *Proc. of the International Symposium on 3D Data Processing, Visualization and Transmission*, pages 1–8, Paris, France, May 2010.

[BBM09] T. Brox, C. Bregler, and J. Malik. Large displacement optical flow. In *Proc. of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 41–48. IEEE, 2009.

[BBPW04] T. Brox, A. Bruhn, N. Papenberg, and J. Weickert. High accuracy optical flow estimation based on a theory for warping. In *Proc. of the European Conference of Computer Vision (ECCV)*, pages 25–36, 2004.

[BCS94] H. Bandelt, Y. Crama, and F. Spieksma. Approximation algorithms for multi-dimensional assignment problems with decomposable costs. *Discrete Applied Mathematics*, 49(1-3):25–50, 1994.

[BETV08] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-up robust features (SURF). *Computer Vision and Image Understanding*, 110(3):346–359, 2008.

[BGPS07] S. Battiato, G. Gallo, G. Puglisi, and S. Scellato. SIFT features tracking for video stabilization. In *Proc. of the International Conference on Image Analysis and Processing*, pages 825–830, 2007.

[BSL$^+$07] S. Baker, D. Scharstein, JP Lewis, S. Roth, M.J. Black, and R. Szeliski. A database and evaluation methodology for optical flow. In *Proc. ICCV*, pages 1–8. IEEE, 2007.

[BTZ96] P. Beardsley, P. Torr, and A. Zisserman. 3D model acquisition from extended image sequences. In *Proc. of the ECCV*, volume 2, pages 683–695. Springer, 1996.

[BWSS09] X. Bai, J. Wang, D. Simons, and Guillermo Sapiro. Video snapcut: robust video object cutout using localized classifiers. *ACM Trans. Graph.*, 28(3):1–11, 2009.

	TV-L^2		TV-L^2 & 2IM		[SLM10]		[SLM10] & 3IM	
	AAE	AEE	AAE	AEE	AAE	AEE	AAE	AEE
art	1.68	10.62	49.45	84.82	1.32	9.34	1.09	8.70
books	11.23	14.60	31.99	55.37	2.67	6.43	1.62	4.85
dolls	1.93	5.81	32.86	67.66	0.53	2.85	0.51	2.27
laundry	7.83	14.16	40.38	58.08	1.27	9.20	1.03	8.39
moebius	0.96	3.67	16.85	23.77	0.87	3.61	0.87	3.13
reindeer	18.89	25.91	18.70	30.50	2.22	16.35	1.02	8.55
waving	2.95	1.03	26.96	31.39	2.74	0.97	2.58	0.92
stonemill	16.72	4.53	48.59	23.16	11.29	3.81	9.73	3.52
RubberW	6.60	0.20	15.07	5.72	6.46	0.20	6.34	0.20
Hydr.	2.98	0.27	9.28	4.39	2.79	0.25	2.77	0.24

Table 2: Including 2-image SURF matching priors (2IM) into TV-L^2 flow significantly increases average angular (AAE) and average endpoint error (AEE) in comparison to the basic TV-L^2 approach. Under consideration of consistency on a loop of three images, inclusion of 3-image SURF matching priors (3IM) decreases the AAE and AEE of the loop consistent TV-L^2 approach [SLM10].

[CS92] Y. Crama and F. C. R. Spieksma. Approximation algorithms for three-dimensional assignment problems with triangle inequalities. *European Journal of Operational Research*, 60(3):273–279, 1992.

[FTV03] V. Ferrari, T. Tuytelaars, and L. Van Gool. Wide-baseline multiple-view correspondences. In *Proc. of the CVRP*, volume 1, pages 718–725, June 2003.

[HP96] A.Y.K. Ho and T.C. Pong. Cooperative fusion of stereo and motion. *Pattern Recognition*, 29(1):121–130, 1996.

[HS88] C. Harris and M. Stephens. A combined corner and edge detector. In *Proc. of the Alvey Vision Conference*, volume 15, pages 147–151, 1988.

[HZ03] R. Hartley and A. Zisserman. *Multiple view geometry in computer vision*. Cambridge University Press, 2003.

[LLM10] C. Linz, C. Lipski, and M. Magnor. Multi-image interpolation based on graph-cuts and symmetric optic flow. In *Proc. of the International Workshop on Vision, Modeling and Visualization*, page to appear, Siegen, Germany, November 2010. Eurographics, Eurographics.

[Low04] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 2(60):91–110, 2004.

[MS05] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *IEEE T-PAMI*, 27(10):1615–1630, 2005.

[MTS+05] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L.V. Gool. A comparison of affine region detectors. *Intern. Journal of Computer Vision*, 65(1):43–72, 2005.

[PS98] C.H. Papadimitriou and K. Steiglitz. *Combinatorial optimization: algorithms and complexity*. Dover Publications, Mineola, New York, USA, 1998.

[RD06] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In *ECCV*, pages 430–443. Springer, May 2006.

[SLM10] A. Sellent, C. Linz, and M. Magnor. Consistent optical flow for stereo video. In *Proc. ICIP*, Sept. 2010.

[SLW+10] T. Stich, C. Linz, C. Wallraven, D. Cunningham, and M. Magnor. Perception-motivated interpolation of image sequences. *ACM Transactions on Applied Perception*, pages 1–28, 2010.

[SP07] D. Scharstein and C. Pal. Learning conditional random fields for stereo. In *Proc. of the CVPR*, pages 1–8. IEEE Computer Society, June 2007.

[Spi00] F.C.R. Spieksma. Multi index assignment problems: complexity, approximation, applications. *Nonlinear Assignment Problems, Algorithms and Applications*, pages 1–12, 2000.

[SS05] K. Shafique and M. Shah. A noniterative greedy algorithm for multiframe point correspondence. *IEEE T-PAMI*, pages 51–65, 2005.

[SSS06] N. Snavely, S.M. Seitz, and R. Szeliski. Photo tourism: exploring photo collections in 3D. *ACM Transactions on Graphics*, 25:835–846, July 2006.

[SZ02] F. Schaffalitzky and A. Zisserman. Multi-view matching for unordered image sets, or "How do I organize my holiday snaps?". In *Proc. of the ECCV*, volume 1, pages 414–431. Springer, May 2002.

[TZ97] P.H.S. Torr and A. Zisserman. Robust parameterization and computation of the trifocal tensor. *Image and Vision Computing*, 15(8):591–605, 1997.

[VRB03] C.J. Veenman, MJT Reinders, and E. Backer. Establishing motion correspondence using extended temporal scope. *Artificial Intelligence*, 145(1-2):227–243, 2003.

[WTP+09] M. Werlberger, W. Trobin, T. Pock, A. Wedel, D. Cremers, and H. Bischof. Anisotropic Huber-L1 optical flow. In *Proc. BMVC*, London, UK, Sept. 2009.

[XJM10] L. Xu, J. Jia, and Y. Matsushita. Motion detail preserving optical flow estimation. In *CVPR*, San Francisco, 2010. IEEE Computer Society.

[YC07] J. Yao and W.K. Cham. Robust multi-view feature matching from multiple unordered views. *Pattern Recognition*, 40(11):3081–3099, 2007.

[YJS06] A. Yilmaz, O. Javed, and M. Shah. Object tracking: A survey. *Computing Surveys*, 38(4):13, 2006.

[Zha94] Z. Zhang. Token tracking in a cluttered scene. *Image and Vision Computing*, 12(2):110–120, 1994.

[ZKP10] C. Zach, M. Klopschitz, and M. Pollefeys. Disambiguating visual relations using loop constraints. In *Proc. of the CVPR*, pages 1–9. IEEE, June 2010.

Accurate Real-Time Multi-Camera Stereo-Matching on the GPU for 3D Reconstruction

Klaus Denker
HTWG Konstanz, Germany
kdenker@htwg-konstanz.de

Georg Umlauf
HTWG Konstanz, Germany
umlauf@htwg-konstanz.de

ABSTRACT

Using multi-camera matching techniques for 3d reconstruction there is usually the trade-off between the quality of the computed depth map and the speed of the computations. Whereas high quality matching methods take several seconds to several minutes to compute a depth map for one set of images, real-time methods achieve only low quality results. In this paper we present a multi-camera matching method that runs in real-time and yields high resolution depth maps.
Our method is based on a novel multi-level combination of normalized cross correlation, deformed matching windows based on the multi-level depth map information, and sub-pixel precise disparity maps. The whole process is implemented completely on the GPU. With this approach we can process four 0.7 megapixel images in 129 milliseconds to a full resolution 3d depth map. Our technique is tailored for the recognition of non-technical shapes, because our target application is face recognition.

Keywords
Stereo-matching, multi-camera, real-time, gpu, computer vision.

1 INTRODUCTION

Stereo matching is a technique to compute depth information of a captured object or environment from two or more 2d camera images. Many applications ranging from remote sensing to robotics, archeology, cultural heritage, reverse engineering, and 3d face recognition [15, 17, 10, 26] use stereo matching. It is the only passive method to generate depth information. This means there is no artificial interaction with the object that might do any harm and only natural light is used for the data acquisition.

The main challenge of stereo matching is the trade-off between the quality of the depth map and the computation time to compute the depth map. For some applications a real-time computation is not important. So many stereo- and multi-view-matching methods focus on high quality results instead of fast computation times. These high quality methods need at least several seconds to compute a single depth map from one set of images [9]. However, for robotics faster computation times are more important than the quality of the depth map. This led to the development of GPU based real-time matching methods [28, 27].

Our target application is 3d face recognition. For face recognition the requirements are somewhere between these fields. A trade-off between a high depth map quality and an acceptable speed must be found. The whole reconstruction and recognition needs to be done in less than half a second. A longer delay is not acceptable for the captured person. Nevertheless, the quality of the reconstructed surface needs to be high enough for a reliable recognition of the person.

1.1 Overview and contribution

In order to classify our approach for the subsequent related work section we give here a brief layout of our system. It is based on weighted normalized cross-correlation for all matching windows of a reference image to a set of additional images from different perspectives. This cross-correlation yields a score for every matching window position and the maximal score indicates the best matching position. This best matching position corresponds to a disparity of the matching windows and thus to the depth information. These steps will be described in Sections 3 - 4. Our contribution in this process is the GPU optimized use of weighted normalized cross-correlations, the combination of multiple cameras to a total score for simultaneously moved matching window, a projection-free depth-map-based deformation of the matching windows, and a sub-pixel precise disparity estimation. These techniques account for the quality of the generated depth maps. To compute the depth maps in real-time our process is implemented on the GPU. This is described in Section 5 and has not be done in such a consequent form before.

2 RELATED WORK

Our method may be classified between two very different classes of stereo matching methods. On the one hand, the high quality methods with long computation time to achieve excellent results. On the other hand,

the fast GPU methods using much simpler algorithms. Therefore, we will contrast our approach to both classes of stereo matching methods.

2.1 High quality methods

High quality stereo matching methods have been developed based on various techniques. The quality of such methods is compared at [19, 21, 25]. Newer benchmark results are available on the associated websites [20, 22, 24].

One of the earliest methods in this class is the adaptive least squares correlation of [6]. In this approach local affine transformations are estimated using a least squares approximation. Although, this method theoretically converges to an optimal solution, the convergence is too slow and the computation too costly due to the size of the linear systems.

Today, best reconstruction quality is achieved by region growing algorithms, e.g. [5, 9]. These methods are typical for high quality matching algorithms, where a set of good matches is generated using a sparse set of interesting features. Then, these good matches are extended with a growing strategy. The growing operations are iterated in combination with filter operations to control the quality of the matches. Because the growing process is based on an optimization of complex objective functions, these methods do not allow a fast GPU implementation.

A novel alternative is the phase only correlation of [23]. Here, the disparity of matching windows is estimated by the phase difference of the image signal along epipolar lines. This requires the computation of a Fourier transformation, which is difficult to implement on the GPU [14]. This is particularly problematic if the Fourier transform must be evaluated for every pixel of the captured image.

Global optimization of a Markov Random Field (MRF) is used in [1]. For each pixel multiple depth hypotheses are stored and the best is picked by the MRF optimization. The solution of this NP-hard problem is approximated using a sequential tree re-weighted message passing algorithm [11]. Although the GPU is used to solve several steps of the algorithm, the global optimization makes it much slower than typical GPU methods.

A particle cloud optimization is used by [8] to generate depth representations for each camera image. The particles are aware of depth discontinuous silhouettes and use a special volumetric view space parametrization instead of the usual image-based parametrization of matching windows. Then, these depth representations are combined and rendered in real-time using the GPU.

Approaches based on dynamic programming, e.g. [12, 18], are relatively similar to our approach. For these methods fields of matching scores are computed for every epipolar line. Within these fields an optimal path is computed using dynamic programming. The computations of the optimal path can either be done on the CPU or on the GPU requiring significant amount of memory.

Our approach is also based on matching scores along epipolar lines, but the computations are local and simple to allow an implementation on the GPU.

2.2 GPU methods

Much faster methods implement the stereo-matching algorithm on the GPU using hardware features of the graphics card like mip-mapping.

A typical example for this class of methods is described in [27]. This approach consists of a set of individual steps of the overall stereo-matching process implemented on the GPU. For the matching score either the sum of squared differences or the sum of absolute differences are used. These matching scores are easily implemented on the GPU, but yield only low quality disparity maps. To exploit the capabilities of the mip-map a pyramidal matching kernel is used, which does not allow for an independent movement of the individual levels in the pyramid. In both aspects our approach improves this method. Some other aspects of [27], like cross-checking and feature aligned matching windows, could easily be integrated to our system.

A different approach of the same first author is [28]. Here five calibrated cameras are matched at once. Using the same technique with a reconfigurable array of 48 cameras is described in [30]. For this technique the matching window covers only one pixel to simplify the computations on the GPUs. This local approach is not stable but very fast and avoids all disadvantages of large matching windows.

Another technique for a large number of images is [29]. It is not as fast as the other GPU methods, but includes a volumetric reconstruction of the objects. A plane sweep method is used for depth estimation on non-rectified images.

The method from [2] uses the pyramidal matching kernel and mip-mapping from [27] and adds a foreground/background separation on the GPU. This additional step avoids typical artifacts of the pyramidal kernel like wrong depth estimates for regions with low texture details usually found in the background. Our improved multi-level approach does not show such problems.

3 THE CAMERA SYSTEMS

We built a system of four USB Logitech® QuickCam® Pro 9000 cameras, see Figure 1(a). Each camera is used at a resolution of 960×720 at five frames per second. The cameras could yield a much higher resolution, but the bandwidth of the USB 2.0 controllers is limited.

To improve the quality for later face recognition, we built a second camera system of four Point Grey Flea®2 FireWire 800 cameras, see Figure 1(b). These cameras synchronously capture images at a resolution of 1392×1032 at 15 fps. For synchronization we use all four cameras on a single FireWire 800 Bus. Thus, in RGB mode only a frame rate of 3.75 fps is possible. This can be improved by de-mosaicking on the GPU and transferring the data in eight bit raw mode. This allows for 11.25 fps.

Our experiments showed that a Y-constellation of four cameras as shown in Figure 1 gives the best results. The image of the central camera is used as reference image for matching and texturing. Each possible image pair has a different angle. Otherwise preferred directions of the camera constellation could deteriorate the detection of features along these directions, e.g. an image containing horizontal stripes causes problems for horizontal camera arrangements.

Independently of the used hardware system, our method can be adapted to other camera constellations. This adaption is much easier for camera systems where all cameras are mounted on a plane perpendicular to the viewing direction. The individual camera images are rectified using a lens correction similar to [4].

4 MATCHING

The overall matching process consists of several nested loops shown in Figure 3. We describe this process from the inner to the outer loop.

4.1 Stereo matching

The aim of stereo matching is to find corresponding points in two images. Usually two square regions, called *matching windows* are compared. These windows are moved over the images to find the best matching position. To identify the best position, a score is computed, that rates the similarity of two matching windows. Similar to [13] we use a *weighted normalized cross-correlation* on RGB color values. First the weighted average color $\overline{f}_i$ of the matching window W_i in the i-th image is computed

$$\overline{f}_i = \sum_{(x,y)\in W_i} w(x,y) f(x,y). \quad (1)$$

Here $w(x,y) = \cos^2(\pi x/a) \cdot \cos^2(\pi y/a)$ is a weight function that smooths the result to emphasize pixels at the center of the matching window over pixels at the border, and a denotes the matching window size in pixels. Then the weighted auto-correlation α_i of each matching window with itself is computed as

$$\alpha_i = \sum_{x,y} w(x,y) \left[f_i(x,y) - \overline{f}_i \right]^2. \quad (2)$$

To evaluate the similarity of two matching windows W_i and W_j the weighted cross-correlation $\beta_{i,j}$ is computed

$$\beta_{i,j} = \sum_{x,y} w(x,y) \left[f_i(x,y) - \overline{f}_i \right] \cdot \left[f_j(x,y) - \overline{f}_j \right]. \quad (3)$$

The weighted normalized score $\gamma_{i,j}$ is computed as the weighted cross-correlation normalized by the geometric mean of the respective weighted auto-correlations

$$\gamma_{i,j} = \beta_{i,j} / \sqrt{\alpha_i \cdot \alpha_j}. \quad (4)$$

4.2 Multi-camera matching

Stereo matching evaluates the similarity of two matching windows. We extend this score to a set of n cameras and matching windows by summing up the weighted normalized scores of all possible image pairs. Thus, we need $n(n-1)/2$ stereo matching operations. To compute a total score we compute a camera score

$$\gamma_i = \sum_{j \neq i} \gamma_{ij} \quad (5)$$

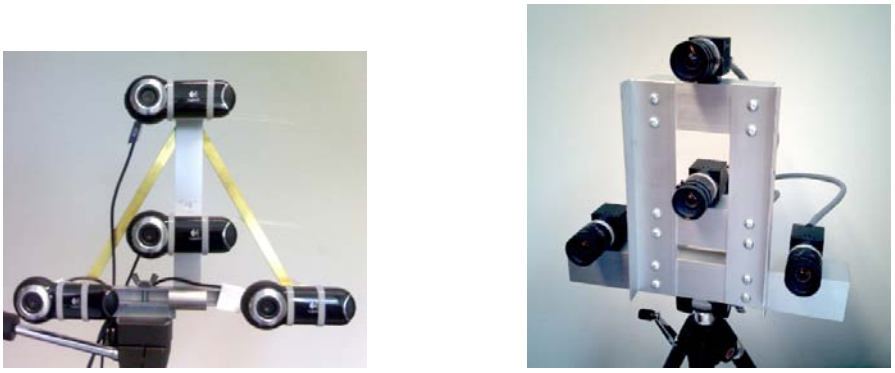
and a total score

$$\gamma = \sum_i \gamma_i - 2 \min_i \gamma_i. \quad (6)$$

This eliminates all scores from the worst matching camera to improve robustness to occlusion on one of the cameras. The total score is used to evaluate the similarity of matching windows of multiple cameras simultaneously.

4.3 Moving the matching windows

Between the images a disparity estimation is computed to get the depth information. Therefore, the matching windows are moved simultaneously over all images. A total score of each position and the best matching window position with the highest total score are computed. Since the evaluation of all possible positions is too expensive, the movement of the matching window is limited to the epipolar lines projected by the center point of the matching window of the reference image. The image of the central camera is used as reference image, i.e. the matching window on the central image is fixed. Figure 2 shows the simultaneous movement of the matching windows in the other images along the epipolar lines. These movements along the epipolar line have a step size of one pixel for our camera configuration. For other camera configurations the step size depends linearly on the distance to the central camera. We test $3 \leq k \leq 35$ different positions for each matching window, see Section 4.5. Note that the color values for the score computations are bi-linearly interpolated to allow an exact movement along the epipolar line. The best similarity of the matching windows is marked by the matching window position with the highest score



(a) USB camera system. (b) FireWire camera system.

Figure 1: For our experiments we use two systems of four cameras arranged in an upside down Y-constellation.

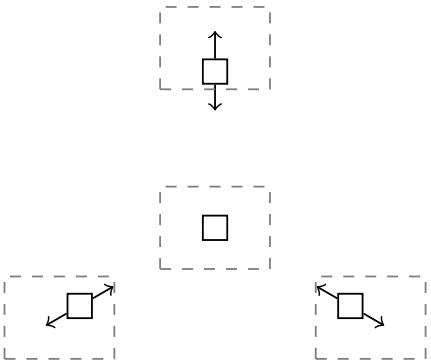


Figure 2: Moving the matching windows (solid squares) in all images (dashed rectangles) along epipolar lines (arrows) simultaneously.

s_{best}. From the position on the epipolar line, the disparity d_{best} of the best match is estimated. The real depth can be computed by reverse projection using the position of the reference camera, the distances to the other cameras, and the disparity.

4.4 Sub-pixel matching

To achieve sub-pixel precision for the disparity map we use a method similar to sub-pixel accurate edge detection of [3]. The best disparity is achieved at a local maximum of the total score, i.e. both neighboring scores s_{left} and s_{right} are smaller or at most one of them is equal to s_{best}

$$s_{\text{left}} \leq s_{\text{best}} > s_{\text{right}} \quad \text{or} \quad s_{\text{left}} < s_{\text{best}} \geq s_{\text{right}}. \qquad (7)$$

Interpolating these three total scores with a quadratic polynomial yields a best sub-pixel score at the global maximum of the quadratic polynomial. This maximum is achieved within half the distance to the neighbor positions. The position of this maximum is the interpolated sub-pixel disparity d_{sub}.

4.5 Multi-level matching

Our method generates disparity data for one image at a fixed resolution. To allow large disparities, many possible matching window positions must be evaluated. Because this is computationally expensive, we use a real multi-level approach that can reduce the effort for large disparities. A similar approach in [27] uses a matching pyramid. In contrast to our method, the windows on different detail levels cannot be moved independently.

Independent levels allow us to re-use high level information to get a much faster low level disparity computation. The graphics card stores the lens corrected image in a mip-map at eight different resolutions. Each level has half the horizontal and vertical resolution of the one below. All matching windows have a fixed size of 7×7 pixels. A smaller window size increases the noise while a larger size blurs sharp features. Starting on the coarsest resolution level $l = 7$, the disparities of all pixels in the reference images are computed at the same coarse resolution. The matching windows are evaluated at $k = 35$ different positions. Then the image resolution is doubled and the same process starts again, while $k = 1 + 2\lfloor 1.5 + l^2/3 \rfloor$ is reduced quadratically. As starting position for the matching windows on lower levels, the bi-linearly interpolated disparities of the next coarser level are used. Thus, the matching window moves k pixels around the best position of the previous level.

4.6 Deformed matching windows

Square matching windows can only yield good results, if the captured object surface is parallel to the image plane. Every surface not parallel to the image plane generates imprecision. To avoid this the matching windows are deformed to fit the perspective deformation of the object surface. The idea is based on [7], but we use the multi-level depth information and a projection free computation.

To estimate the deformation we use the disparity map of the previous multi-level step. First nine disparity values at the corners, the edge midpoints, and the center of the matching window are interpolated. This gives a disparity estimate for every pixel in the actual matching

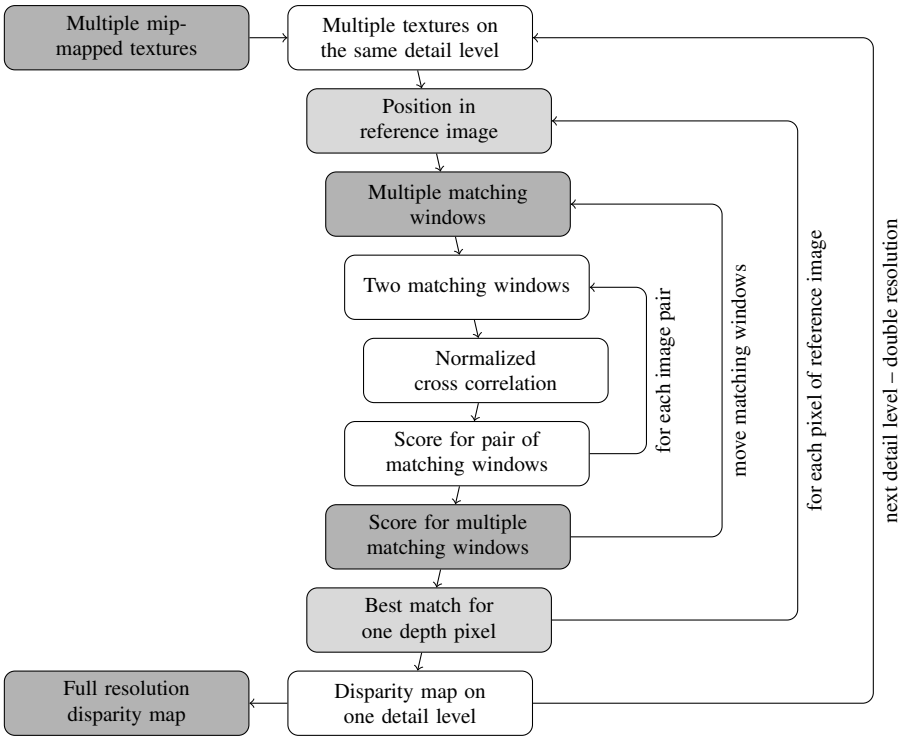


Figure 3: Overview of our matching process.

window. Subtracting the disparity at the center of the matching window yields a local displacement for every pixel. This displacement is added to the pixel coordinates before the color values are read. This results in a matching window adapted to the perspective of the previous level without computing any perspective projections. Note, that for planar object surfaces this approach is almost equivalent to the projections used by [7]. The difference is that it is based on disparity instead of depth.

4.7 Measuring the matching quality

For each resolution level a complete disparity map is computed. So, for each pixel of this map the best total score computed is also stored. Averaging these total scores over multiple resolution levels gives a quality measure for each pixel of the full resolution depth map, see Figure 4(b). Pixels with low quality measures can be masked for rendering or subsequent computations of the user application.

The quality measure is also used to improve the performance of the multi-level matching. A low quality measure on a coarse matching step usually causes the finer level matches in this region to fail too. Matching calculations are skipped if the quality measure on the next coarser level is too low.

5 IMPLEMENTATION ON THE GPU

The method described so far uses images and generates a depth image as result. Therefore, we use GLSL fragment shaders for the GPU implementation. A fragment shader is a program that runs in parallel on the GPU and processes one or multiple texture images into one result image. For our shader operations we need GPUs which support at least shader model 4.0. The required amount of computations in a single shader run is not feasible on older GPUs.

5.1 GPU lens correction

Our input data are multiple raw camera images. Each raw image is corrected by a shader implementing a lens correction similar to [4]. The resulting corrected images are rendered into separate textures. Each of these textures is then transformed into a mip-map. These mip-maps of the corrected images are used by all subsequent shaders of our system.

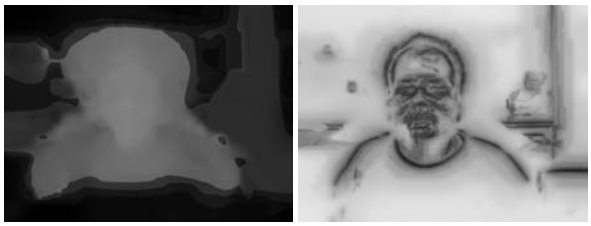
5.2 GPU optimized matching

A single pixel shader run usually computes the color values for one result image, each pixel separately. More complex computations require the combination of multiple shader runs. Three fragment shader programs are used for each step of our multi-level matching.

The first shader takes the corrected image mip-map and computes the weighted average color of the pixels of a matching window at the actual resolution level. These averages are rendered to separate average textures. This shader is invoked once for every image.

The second shader takes the corrected image mip-map and the average texture and computes the weighted auto-correlation for the same matching window. Again

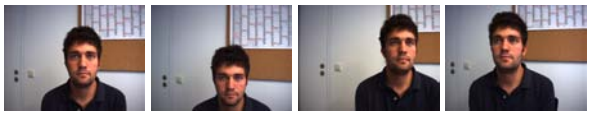
(a) The four captured sample images.



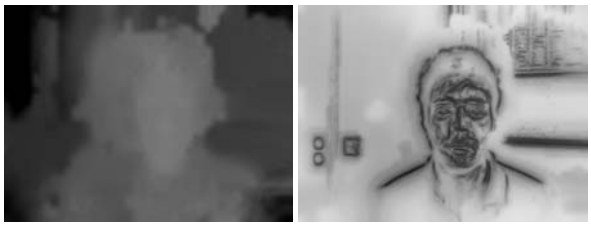
(b) Result textures. Disparity map (left) and quality measure (right).

(c) Reconstructed 3d model.

Figure 4: Example from our USB camera system.



(a) The four captured sample images.



(b) Result textures. Disparity map (left) and quality measure (right).

(c) Reconstructed 3d model.

Figure 5: Example from our FireWire camera system.

the result is rendered to a separate auto-correlation texture and the shader is invoked once for every image.

The third shader takes the average and auto-correlation textures and performs all matching operations, i.e. it moves the deformed matching windows, computes the total score, and finds the best sub-pixel score. The result is rendered as the disparity map, the best total score of the finest resolution and the quality measure to the three color channels of a separate texture. These three shaders are invoked once per resolution level.

Most important strategies used to improve the GPU performance are the pre-calculation of weighted average and weighted auto-correlation just described and the multi-level matching described in Section 4.5.

6 RESULTS

Our target application is face recognition. We present our results in that area. For easier comparison with other algorithms we also applied our algorithm to a well known benchmark for stereo matching.

6.1 Face reconstruction

We took some example images with our USB camera system shown in Figure 4(a). The disparities between these images are very large. The result texture of the fragment shaders holds the disparity map, the best total score of the finest resolution level, and the quality measure encoded in the color channels, see Figure 4(b).

After transformation of the disparities to depth values, the data can be rendered as 3d model, see Figure 4(c). The low quality regions are masked and ignored in this rendering.

A typical problem of stereo matching can be seen at the highlights on the forehead generating small dents, because the reflection is further away from the cameras than the forehead. More diffuse lighting could avoid this problem. The computation for the example images takes an average processing time of 129 ms on an NVidia GeForce GTX 285 GPU. This allows real-time frame rates of 7.5 fps.

A higher resolution of 1392×1032 is achieved by the FireWire camera system. An example image set is shown in Figure 5(a). Figure 5(b) shows the result textures and Figure 5(c) a 3d model of the resulting depth

map. The higher camera resolution yields a better shape quality at the most important regions of the face. Especially the reconstruction of the eye and mouth regions is much more precise.

For this example an average processing time of 263 ms is needed on the same GPU. For images of 30 different persons we get an average processing time of 254 ms. In most of these images the face region is smaller than in the displayed examples, so the computations are a bit faster. In comparison to the first example, the computation time grows almost linearly with the number of pixels p. This conforms to a runtime of $O(p \log p)$ for our multi-level algorithm: The matching window size, the stretch of the window movement, and the count of image pairs are constant. So the worst case costs for the computations in each depth map pixel is constant. The pixels of the resulting depth map, or smaller versions of it, are computed once for each of the $\log_2(\text{width}) \in O(\log p)$ multi-level steps. Hence the overall count of pixel calculations and the complexity of the algorithm is within $O(p \log p)$.

6.2 Stereo vision benchmarks

Several benchmarks can be used to compare the quality of stereo matching algorithms [19, 21, 25]. Our algorithm is tailored to face reconstruction and contains simplifications that require a planar camera configuration. Thus, it is not comparable to the benchmark [25]. Furthermore, our algorithm is also tailored to large disparities between the images and achieves a much better reconstruction quality using more than two cameras. So, only a comparison with the results of the extended datasets of the Middlebury stereo benchmark [20] is relatively fair. However, this benchmark does not provide an official score.

Compared to the algorithms providing results and timings for these benchmark our algorithm works much faster. At the same time the quality of our result is comparable to the quality of these algorithms. However, for this comparison we have to adapt our algorithm.

For the Middlebury stereo evaluation [20], we integrated a modified local version of Multi Hypothesis Matching [1] to improve the sharpness of edges in our algorithm. The movement range of the matching windows is extended to the depth extrema of the local neighborhood on the last detail level. Instead of evaluating only the best matching score, the eight best matching scores are stored. A post-processing step re-weights these scores based on the values and depth distances to the best scores in the direct pixel neighborhood. The re-weighting is repeated two times without any global optimization as in [1]. This multi hypothesis matching is implemented as an post-processing fragment shader on the GPU. The additional shader and the increased range for the matching windows cause a large performance loss. Processing our example images at a resolution of 960×720 pixels takes 900 ms. This is still faster than the other algorithms in [20], but not fast enough for our target application.

Figure 6 shows our algorithm applied to the extended *Tsukuba* dataset from [20, 16]. The two images in Figure 6(b) show the results from all five input images without and with the additional edge improvement.

7 CONCLUSION AND FUTURE WORK

The quality of the resulting surface model is sufficient and the processing times are more than sufficient for our target application 3d face recognition. Additional methods like cross-checking that can be implemented on the GPU could further improve our results. Furthermore, for an application of our method in a face recognition system, a simple method to guide persons to the optimal distance from the camera system is required.

For the future we plan to record synchronous video sequences with the FireWire camera system. Similar to multi-level matching, the matching information of an earlier video frame could be used to improve the performance.

ACKNOWLEDGMENTS

This work was supported by DFG GK 1131 and AiF ZIM Project KF 2372101SS9. We thank Jens Hensler for his help on creating a collection of face pictures.

REFERENCES

[1] Neill D. Campbell, George Vogiatzis, Carlos Hernández, and Roberto Cipolla. Using multiple hypotheses to improve depth-maps for multi-view stereo. In *Proc. of the 10th European Conf. on Computer Vision*, pages 766–779, 2008.

[2] Jia-Ching Cheng and Shin-Jang Feng. A real-time multiresolutional stereo matching algorithm. In *ICIP (3)*, pages 373–376, 2005.

[3] F. Devernay. A non-maxima suppression method for edge detection with sub-pixel accuracy. Technical Report RR-2724, INRIA, 1995.

[4] F. Devernay and O. Faugeras. Straight lines have to be straight: automatic calibration and removal of distortion from scenes of structured enviroments. *Mach. Vision Appl.*, 13(1):14–24, 2001.

[5] Yasutaka Furukawa and Jean Ponce. Accurate, dense, and robust multi-view stereopsis. In *CVPR*, pages 1–8, 2007.

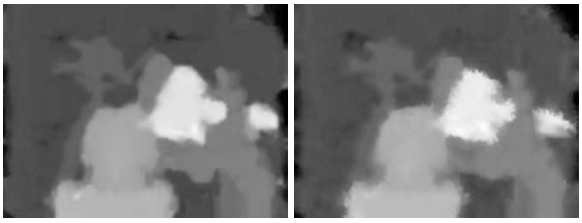
[6] A W Gruen. Adaptive least squares correlation: A powerful image matching technique. *South African Journal of Photogrammetry, Remote Sensing and Cartography*, 14:175–187, 1985.

[7] Hiroshi Hattori and Atsuto Maki. Stereo matching with direct surface orientation recovery. In *In Ninth British Machine Vision Conference*, pages 356–366, 1998.

[8] Alexander Hornung and Leif Kobbelt. Interactive pixel-accurate free viewpoint rendering from images with silhouette aware sampling. *Comp. Graph. Forum*, 28(8):2090–2103, 2009.

[9] Andreas Klaus, Mario Sormann, and Konrad Karner. Segment-based stereo matching using belief propagation and a self-adapting dissimilarity measure. In *Proc. of the 18th Int. Conf. on Pattern Recognition*, pages 15–18, 2006.

(a) The extended *Tsukuba* dataset pictures.



(b) Result disparity map of our algorithm without (left) and with edge enhancement (right).

(c) Ground truth disparity map (left) and 3d rendering of the result with edge enhancement (right).

Figure 6: Results of the extended *Tsukuba* dataset from the Middlebury stereo benchmark [20].

[10] Reinhard Koch, Marc Pollefeys, and Luc Van Gool. Realistic 3-d scene modeling from uncalibrated image sequences. In *ICIP'99, Kobe: Japan*, pages 500–504, 1999.

[11] Vladimir Kolmogorov. Convergent tree-reweighted message passing for energy minimization. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(10):1568–1583, 2006.

[12] Cheng Lei, Jason Selzer, and Yee-Hong Yang. Region-tree based stereo using dynamic programming optimization. In *Proc. of the 2006 IEEE Conf. on Computer Vision and Pattern Recognition*, pages 2378–2385, 2006.

[13] J. P. Lewis. Fast template matching. In *Vision Interface*, pages 120–123, 1995.

[14] Kenneth Moreland and Edward Angel. The FFT on a GPU. In *Proc. of the ACM Conf. on Graphics Hardware*, pages 112–119, 2003.

[15] Don Murray and Jim Little. Using real-time stereo vision for mobile robot navigation. In *Autonomous Robots*, pages 161–171, 2000.

[16] Yuichi Nakamura, Tomohiko Matsuura, Kiyohide Satoh, and Yuichi Ohta. Occlusion detectable stereo – occlusion patterns in camera matrix. In *CVPR*, pages 371–378, 1996.

[17] D.T. Pham and L.C. Hieu. Reverse engineering - hardware and software. In V. Raja and K.J. Fernandes, editors, *Reverse Engineering - An Industrial Perspective*, pages 33–30. Springer, 2008.

[18] H. Sadeghi, P. Moallem, and S. A. Monadjemi. Feature based dense stereo matching using dynamic programming and color. *International Journal of Computational Intelligence*, 4(3):179–186, 2008.

[19] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *Int. J. Comput. Vision*, 47(1-3):7–42, 2002.

[20] D. Scharstein and R. Szeliski. Middlebury stereo vision page. `http://vision.middlebury.edu/stereo/`, 2007.

[21] S.M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *Proc. of the 2006 IEEE Conf. on Computer Vision and Pattern Recognition*, pages 519–528, 2006.

[22] Steve Seitz, Brian Curless, James Diebel, Daniel Scharstein, and Richard Szeliski. Multi-view stereo. `http://vision.middlebury.edu/mview/`, 2009.

[23] T. Shibahara, T. Aoki, H. Nakajima, and K. Kobayashi. A sub-pixel stereo correspondence technique based on 1d phase-only correlation. In *ICIP07*, pages 221–224, 2007.

[24] Christoph Strecha. Multi-view stereo test images. `http://cvlab.epfl.ch/~strecha/multiview/`, 2008.

[25] Christoph Strecha, Wolfgang von Hansen, Luc J. Van Gool, Pascal Fua, and Ulrich Thoennessen. On benchmarking camera calibration and multi-view stereo for high resolution imagery. In *CVPR*, pages 1–8. IEEE Computer Society, 2008.

[26] Francesca Voltolini, Sabry El-Hakim, Fabio Remondino, and Lorenzo Gonzo. Effective high resolution 3d geometric reconstruction of heritage and archaeological sites from images. In *Proc. of the 35th CAA Conference*, pages 43–50, 2007.

[27] Ruigang Yang and Marc Pollefeys. A versatile stereo implementation on commodity graphics hardware. *Real-Time Imaging*, 11(1):7–18, 2005.

[28] Ruigang Yang, Greg Welch, and Gary Bishop. Real-time consensus-based scene reconstruction using commodity graphics hardware. In *Proc. of the 10th Pacific Conf. on Computer Graphics and Applications*, pages 225–235, 2002.

[29] Christopher Zach, Mario Sormann, and Konrad F. Karner. High-performance multi-view reconstruction. In *3DPVT*, pages 113–120, 2006.

[30] Cha Zhang and Tsuhan Chen. A self-reconfigurable camera array. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Sketches*, page 151, 2004.

Ambient Occlusion Opacity Mapping for Visualization of Internal Molecular Structure

David Borland
The Renaissance Computing
Institute, USA
borland@renci.org

ABSTRACT

Molecular surfaces often exhibit a complicated interior structure that is not fully visible from exterior viewpoints due to occlusion. In many cases this interior cavity is the most important feature of the surface. Applying standard blended transparency can reveal some of the hidden structure, but often results in confusion due to impaired surface-shape perception. We present ambient occlusion opacity mapping (AOOM), a novel visualization technique developed to improve understanding of the interior of molecular structures. Ambient occlusion is a shading method used in computer graphics that approximates complex shadows from an ambient light source by rendering objects darker when surrounded by other objects. Although ambient occlusion has previously been applied in molecular visualization to better understand surface shape, we instead use ambient occlusion information to determine a variable opacity at each point on the molecular surface. In this manner, AOOM enables rendering interior structures more opaque than outer structures, displaying the inner surface of interest more effectively than with constant-opacity blending. Furthermore, AOOM works for cases not handled by previous cavity-extraction techniques. This work has been driven by collaborators studying enzyme-ligand interactions, in which the active site of the enzyme is typically formed as a cavity in the molecular surface. In this paper we describe the AOOM technique and extensions, using visualization of the active site of enzymes as the driving problem.

Keywords: Molecular visualization, ambient occlusion, transparent surfaces

1 INTRODUCTION

Visualization has become an essential tool for many scientists working with molecular data. Ball-and-stick, ribbon, and surface renderings are all visualization techniques that enable improved understanding of molecular structures [10, 23, 30, 40]. Our collaborators use visualization techniques such as these to study enzyme-ligand interactions.

Enzymes are proteins that catalyze the transformation of a substrate molecule into a product. The substrate/product is often referred to as the "ligand" with which the enzyme binds. Our collaborators use molecular surface renderings to understand the shape of the active site of the enzyme and its spatial relationship to the ligand during binding. The active site is typically a complicated internal cavity that is largely hidden from exterior viewpoints due to occlusion (Figure 2). Two tools commonly used for viewing occluded structures are transparency and clipping planes.

Applying standard blended transparency to the surface reveals some of the internal structure, but often results in impaired surface-shape perception [24]. Clipping planes can also reveal internal structure, but are typically insufficient for displaying complicated non-planar surfaces such as enzyme active sites. Another potential technique that might be used is applying depth-peeling to remove the closest surface to the viewpoint. Such a technique would be inaccurate for this application, however, as the visible portions of the cavity would be removed, and the second surface may not correspond to the cavity in areas of folds and bumps of the outer surface. To enable improved visualization of the active sites of enzymes, we have developed a technique that uses ambient occlusion information to identify and emphasize these hidden structures.

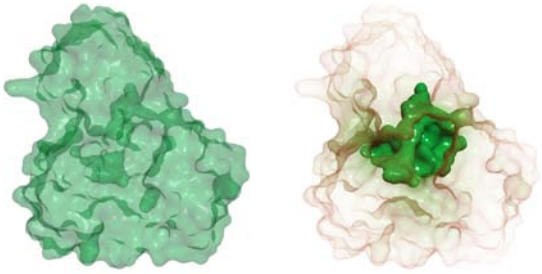


Figure 1: Standard transparency on the left versus ambient occlusion opacity mapping (AOOM) on the right. The structure of the inner cavity, and its context within the outer surface, is easier to understand with AOOM.

Ambient occlusion is a shading method used in computer graphics that approximates complex shadows from an ambient light source. Surfaces surrounded by objects that block ambient light are rendered

darker than those open to the environment, so ambient occlusion is therefore a measure of the "hiddenness" of an object. Ambient occlusion has been successfully used in molecular visualization to help understand surface shape and identify the locations of cavities in molecular surfaces [39]. Instead of using ambient occlusion information solely for enhancing surface shading, ambient occlusion opacity mapping (AOOM) uses ambient occlusion to calculate a variable opacity at each point on the molecular surface. Because ambient occlusion is a measure of the hiddenness of an object, AOOM can render outer structures more transparent and inner structures more opaque, displaying the inner surface of interest more effectively than with standard transparency (Figure 1). In this paper we apply AOOM to the visualization of enzyme active sites and describe various extensions to the core AOOM technique driven by collaboration with biochemists.

The paper is organized as follows: Section 2 provides background information on the biochemistry our collaborators are studying. Section 3 provides related work in the areas of molecular surface visualization, occlusion and transparency in visualization, and ambient occlusion. Section 4 describes the AOOM implementation, extensions, and supplemental visualization techniques. Section 5 concludes and provides future work.

2 SCIENTIFIC BACKGROUND

Our collaborators study the architecture of the active sites of enzymes involved in the tetrapyrrole biosynthesis pathway. The enzymes in this pathway catalyze reactions involving ligands that are necessary for the formation of various molecules such as hemoglobin, vitamin coenzyme B_{12}, and chlorophyll. Of interest are how different enzyme active-site architectures interact with their respective ligands.

PyMOL, an open-source molecular visualization system (www.pymol.org), is used by our collaborators to visualize ligands bound in the enzyme active sites, with crystallographic data taken from the Protein Data Bank (www.pdb.org). Understanding the active site where the ligand binds with the enzyme is especially important for answering questions such as: a) How much space is available for the ligand within the volume occupied by the protein? b) How does the ligand access the active site cavity? c) How completely is the active site cavity filled by the ligand? and d) Which residues in the cavity are close enough to the ligand to provide the anchoring interactions that bind it in place?

Each of these questions involves understanding the shape of the active site cavity, which can be problematic due to self-occlusion of the inner cavity by the outer surface (Figure 2). Dealing with occluded surfaces has long been an area of research in visualization. The next section provides previous work on molecular surface visualization, visualizing occluded surfaces, and ambient occlusion.

Figure 2: The surface cavity forming the active site of the PGB deaminase enzyme is circled. Much of the cavity is occluded by the outer surface.

3 PREVIOUS WORK

3.1 Molecular Surface Visualization

Molecular surfaces are a common visualization technique for studying molecular structures. The solvent-excluded molecular surface is formed by rolling a spherical probe over spheres representing the atoms of the molecule [34]. It represents areas accessible by molecules of a given probe radius. Connolly described methods for generating these surfaces [8, 9]. Methods improving the efficiency and quality of computing these surfaces have also been described [1, 6, 36, 41]. While such methods for producing molecular surfaces are necessary for the visualizations produced in this paper, they do not address the problem of visualizing the occluded interior structure of the generated surfaces.

The problem of visualizing protein docking using surfaces has been addressed [27]. This approach computes the intermolecular negative volume of two docked proteins to determine the amount of intersection between the two surfaces, with the purpose of enhancing drug-design by testing different potential conformations. While effective for such work, this approach is not directly applicable to the biochemistry presented in this paper, which involves data with known structure and no surface intersection.

Methods for analytically extracting pockets and cavities using computational geometry techniques also exist. CASTp uses the weighted Delaunay triangulation and alpha complex to identify and measure the area and volume of pockets and cavities [17], and is available as a PyMOL plugin. The ability to extract measurements of pockets and cavities is very useful, however comparison with an AOOM rendering demonstrates that important features of the cavity may be missed, such as the circled portion of the cavity on the left and the circled access tunnel on the right in Figure 3, Bottom Left (Bottom Right includes a Focal Region technique dis-

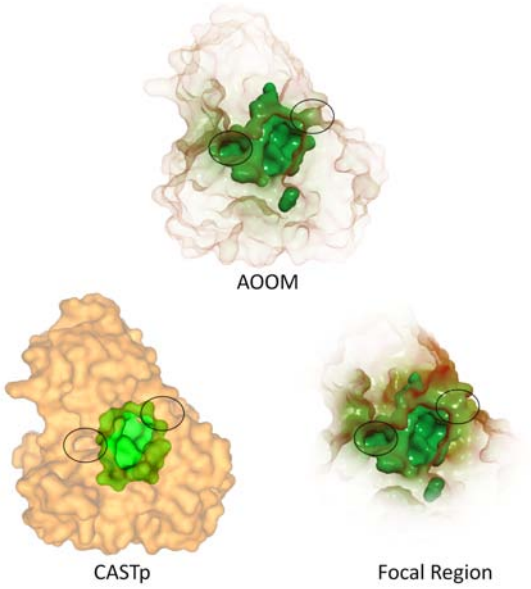


Figure 3: CASTp cavity extraction (bottom left, rendered using PyMOL) does not extract the entire cavity, and does not conform to the original molecular surface (missing circled regions). A focal-region approach (bottom right) based on distance from the center is less effective than AOOM (top), as it occludes portions of the inner cavity (circled access tunnel on the right) and erodes regions of the outer surface that otherwise provide visual context.

cussed in Section 3.2), because the cavity is not calculated from the full molecular surface, but instead from the extracted atoms that form the cavity. Also, there are a number of structures for which CASTp fails, whereas AOOM will work for any molecular surface (Figure 4). Future work will include augmenting AOOM with the types of analytical capabilities provided by CASTp. A promising step in this direction involves extracting the cavity by thresholding based on ambient occlusion information, followed by connected components analysis to remove smaller pockets in the surface (Figure 4, Bottom).

Recent work has resulted in techniques for producing simplified abstractions of complicated molecular surfaces [7]. While this technique does not directly address revealing hidden internal structure, it may prove beneficial to combine AOOM with such abstracted surfaces, as AOOM will work with any surface-based representation. Furthermore, AOOM could be applied to other representations, such as ball-and-stick and ribbon renderings.

3.2 Occlusion and Transparency

Occlusion is the most powerful of all depth cues [43]. However, occlusion can be problematic when visualizing 3D data, as objects of interest can be hidden from view. Applying transparency to occluding objects can

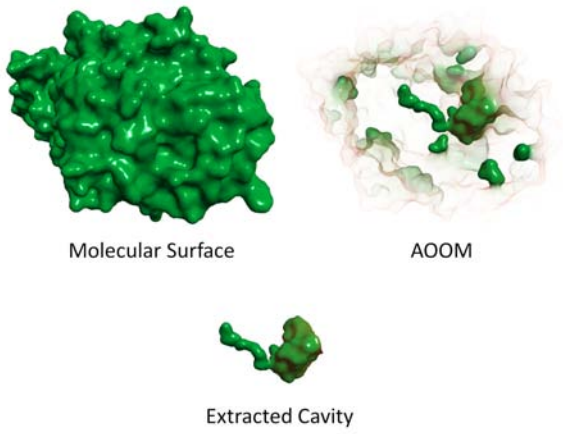


Figure 4: A molecular structure [35] (top left) for which CASTp fails, that reveals a long tube-like cavity structure when rendered using AOOM (top right). If desired, the cavity can be extracted by thresholding on ambient occlusion information followed by connected components analysis (bottom).

make hidden objects visible, but simple transparent surfaces do a poor job of conveying surface shape [24]. Various techniques have therefore been developed to enable more effective visualization of occluding and occluded surfaces.

Illustrative Techniques Illustrative techniques include a surface-rendering technique for view-dependent tranparency that aims to automatically produce transparent surfaces similar to technical illustrations [15]. Later work describes techniques for automatically producing breakaway and cutaway illustrations of nested surfaces [16]. These techniques are useful for nested surfaces, but do not address a single self-occluding surface. Similar work has been applied to isosurfaces extracted from volumetric data [19], which is useful for objects within the volume of the isosurface, but not for a single self-occluding surface.

Focus-and-Context Techniques A class of direct volume rendering techniques has been developed to reveal the inner structure of volumes while retaining some outer structure to maintain context. Importance-driven volume rendering highlights presegmented regions based on user-supplied importance criteria [42]. Opacity reduction of occluding volumes by finding volumetric features indicating a separation between areas with similar voxel intensities has been described [2, 3]. Selective opacity reduction of regions using a function of shading intensity, gradient magnitude, distance to the eye point, and previously accumulated opacity has also been described [5]. Opacity-peeling can be used to remove some fixed number of fully-opaque layers of material [32]. These focus-and-context techniques use various features of the volumetric data to modulate opacity and reveal hidden structure, and therefore

do not apply directly to molecular surface rendering. A depth-dependent focal region can also be used for opacity modulation [29]. However, applying a similar technique to our data shows that it is insufficient by itself due to the irregular geometries formed by molecular surfaces, even for a relatively round and centralized cavity (Figure 3, Bottom Right). The work of [11] is closest to that described in this paper, as ambient occlusion is also used for modulating opacity. However, their work focuses on volumetric data, and does not perform the smoothing process described in 4.2 (Figure 6 shows AOOM without smoothing).

3.3 Ambient Occlusion

For the techniques listed above, some method of determining the object or volume of interest is necessary, either via distinct and separate surfaces or via functions of the underlying volume. For displaying the inner cavity of a molecular surface, we need a means to determine which portions of the surface constitute the inner cavity of interest. To do so we calculate ambient occlusion for the surface.

Ambient occlusion [4, 26] is a shading method used in computer graphics that approximates complex shadows from an ambient light source by rendering objects darker when surrounded by other objects (Figure 5, Left). The basic algorithm involves casting a number of rays at various angles from each point on a surface, keeping track of the number of rays that intersect another (or the same) surface. Recent work has focused on real-time ambient-occlusion calculation for dynamic scenes [20, 25, 37, 38], but for our static surfaces it is sufficient to use a pre-calculated ray-tracing approach and store the ambient occlusion per-vertex. The ambient occlusion term O_p at a point p on a surface with normal N can be computed by integrating the visibility function V_p over the hemisphere Ω with respect to the projected solid angle:

$$O_p = \frac{1}{\pi} \int_{\Omega} V_p(\vec{\omega})(N \cdot \vec{\omega})\, d\omega, \tag{1}$$

where $V_p(\vec{\omega})$ is zero if p is occluded in the direction $\vec{\omega}$, and one otherwise. The dot product $N \cdot \vec{\omega}$ results in a cosine-weighting across the hemisphere. Using a cosine-weighted distribution of rays therefore removes the need for this cosine factor, resulting in a simple ratio of the number of rays that intersect a surface r_i and the number of total rays r_t:

$$O_p = \frac{r_i}{r_t}. \tag{2}$$

Areas of the surface that are largely occluded will therefore have a high O_p value.

Ambient occlusion rendering was developed to enhance realism in computer graphics by replacing the standard ambient term by $1 - O_p$ to darken objects blocked from ambient light. Ambient occlusion has also proven useful for scientific visualization. For example, with molecular surface rendering, the locations of cavities in the molecular surface become more apparent (Figure 5, Left). Because ambient occlusion is a measure of the "hiddenness" of a particular point on a surface, we can use ambient occlusion information to identify hidden structures and render them more opaquely to provide visual emphasis.

4 AMBIENT OCCLUSION OPACITY MAPPING

Ambient occlusion opacity mapping (AOOM) uses ambient occlusion information to modulate the opacity of the molecular surface. Areas with high ambient occlusion that would typically be rendered dark, are instead rendered more opaque than areas with low ambient occlusion. The color of the surface can also be adjusted based on ambient occlusion to enhance perception of the inner cavity versus the outer surface. This section describes the implemenation details of AOOM, extensions to the core AOOM technique enabling enhanced opacity control, and supplemental visualization techniques used with AOOM to visualize enzyme-ligand interactions.

4.1 Implementation Details

The examples shown here use surfaces exported from PyMOL. A molecular surface with a probe size of 1.4 Å ($\approx$ radius of water) is used. For most of the examples in this paper, we show PBG deaminase [28] for consistent comparison. Figures 4 and 12 show AOOM applied to other molecules.

We have implemented AOOM via extensions to the Visualization Toolkit (VTK) (www.vtk.org). Surfaces exported from PyMOL are loaded in OBJ or VRML format. Ambient occlusion is pre-computed on the CPU for each vertex on the input via a ray-casting approach, and stored as scalar point data. As with other ray-tracing techniques, calculating per-vertex ambient occlusion is embarrassingly parallel, so we accelerate computation by distributing computation across processing units. Results are typically stored in one of VTK's polygonal file formats so that computation is only necessary once. Because the ambient occlusion is pre-computed, there is a neglible decrease in performance when rendering with AOOM.

Depth sorting is performed to obtain correct blending, which can affect performance for large surfaces, however this is also the case for standard transparency. A depth-peeling approach could also be used to achieve order-independent transparency [14, 18]. Because the O_p term constitutes a scalar field mapped to the surface, a full range of color and opacity functions can be applied, typically in the same fragment program used for lighting and shading. In the simplest case, the opacity

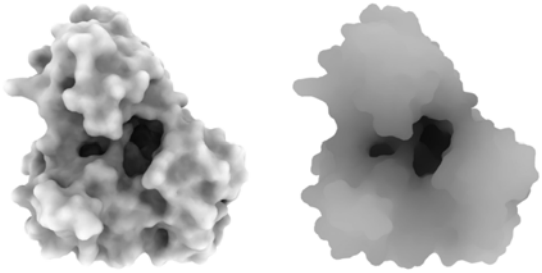


Figure 5: Ambient occlusion (left) vs. smoothed ambient occlusion (right). Using smoothed ambient occlusion for opacity enables filtering of small-scale concavities in the surface.

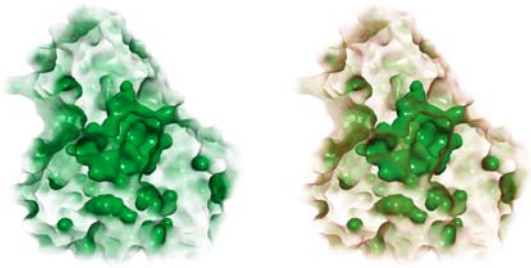


Figure 6: The simplest implementation of AOOM, in which ambient occlusion is directly mapped to opacity. The image on the left uses a constant color, and the image on the right applies a color map to the ambient occlusion.

is set equal to the O_p ambient occlusion term, $\alpha = O_p$, via a linear lookup table, and a constant color is used (Figure 6, Left). A double-ended color map that separates high and low ambient occlusion values can also be applied to help viewers distinguish interior and exterior structures (Figure 6, Right). For the examples in this paper we apply a color map from orange (low ambient occlusion) to green (high ambient occlusion).

This approach is more effective than standard transparency (Figure 1, Left) in revealing the structure of the inner cavity, but can be improved upon with additional opacity controls.

4.2 Smoothed Ambient Occlusion

Although Figure 6 demonstrates an improvement over standard transparency techniques, there are still areas of the outer surface that occlude the interior cavity, due to small concave pockets formed on the surface. To deemphasize these pockets, we smooth the ambient occlusion data over the surface. Smoothing the ambient occlusion filters out small-scale features, while retaining the larger cavity (Figure 5). We smooth the ambient occlusion data directly on the surface by iteratively solving the diffusion partial differential equation:

$$\frac{\partial u}{\partial t} = D\,\nabla^2 u, \tag{3}$$

where D is a constant controlling the amount of diffusion per time step ($= 1$ in the general case). This approach is equivalent to smoothing using a Gaussian filter, with more iterations equal to a Gaussian with a larger standard deviation. We solve the diffusion equation rather than performing direct convolution with a Gaussian because solving the diffusion equation iteratively only requires sampling immediate neighbors in the polygonal mesh. In practice we have found that 100 iterations works well for our data, and has been used for all images. The smoothing is typically performed once at run-time, upon loading the data.

Although smoothed ambient occlusion is useful for selecting the scale of features that are rendered more opaquely, we also provide the ability to use the original ambient occlusion for color to retain detail (Figure 8).

4.3 Opacity Control

Arbitrary functions can be used to map the smoothed ambient occlusion values to opacity, however we desire a mapping that maintains the opacity of the inner cavity while providing control over the opacity of the outer surface. We experimented with functions such as *smoothstep*, however the following equation was determined via visual inspection to produce better results:

$$\alpha = \left(\frac{O_p}{\tau}\right)^{\rho}, \tag{4}$$

where α is clamped to $[0,1]$. The τ parameter provides a threshold such that an $O_p >= \tau$ gives an α of 1. A τ of 0.7 is used for all images in this paper (other than Figure 6, which uses a simple linear ramp). The ρ parameter controls the shape of the curve as an exponential, and in practice is allowed to vary over $[0,10]$. For a τ value of 1, a ρ value of 1 will give the same result as the simple linear opacity mapping decribed in section 4.1. Increasing ρ from 1 will reduce the opacity of the outer surface to a greater degree than the inner surface. Decreasing ρ from 1 will increase the overall opacity until ρ reaches 0, resulting in a constant α of 1 and a fully opaque surface. A graphical representation of Equation 4 is shown in Figure 7. The effect of changing ρ is shown in Figure 8, and is typically adjusted interactively by the user.

4.4 Supplementary Visualization Techniques

We also apply a number of supplementary visualization techniques to enable improved understanding of the enzyme active site cavity and ligand.

Silhouette-Edge Highlighting To maintain contextual information of the outer surface while rendering the interior surface more visible, silhouette-edge highlighting can optionally be applied:

$$\alpha = {\alpha_{in}}^{(\hat{N}\cdot\hat{E}+1)}, \tag{5}$$

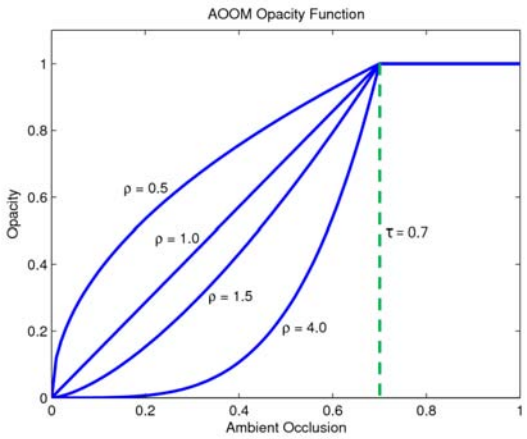


Figure 7: Example AOOM opacity functions.

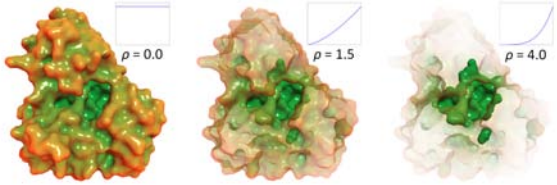


Figure 8: Result of changing the ρ parameter to adjust outer opacity while maintaining the opacity of the inner cavity.

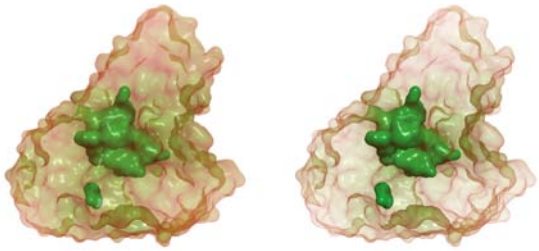


Figure 9: AOOM example without (left) and with (right) silhouette-edge highlighting.

where α_{in} is the opacity after applying Equation 4, and $\hat{N} \cdot \hat{E}$ is the dot product of the surface normal and the eye vector. This equation selectively reduces the opacity of areas on the surface with low opacity that face the viewer. Areas of high opacity are less affected, and areas of full opacity are unaffected. The calculation is performed in the same fragment program used for lighting and AOOM. Other edge highlighting techniques, such as suggestive contours [12, 13], could also be applied. Figure 9 shows the result of applying silhouette-edge highlighting.

Colored Surfaces To better understand the chemistry occuring in the active site, it can be useful to color the molecular surface by atom type. A nominal color coding is employed, with charged residues colored blue for cationic (positive) species and red for anionic (negative) species. The carbon backbone is colored green in our examples (coloring of specific groups, such as yellow sulfur groups, are also used). Because the molecular surface color conveys information, color mapping as described in Section 4.1 is not desirable in this case, as the nominal color encoding will be distorted. The silhouette-edge highlighting described in above can therefore be especially helpful when using such a color coding (Figure 10).

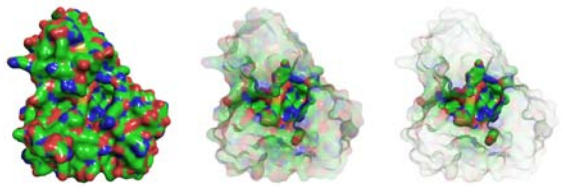


Figure 10: Colored molecular surface (left), along with AOOM renderings without (middle) and with (right) silhouette-edge highlighting.

Enclosed Region Removal Sometimes fully-enclosed regions are formed during the molecular surface calculation. These regions are not accessible from positions exterior to the enzyme (for molecules with a radius >= the surface probe radius) and can add visual clutter to the scene. These regions can be automatically removed by computing connected components on the molecular surface and rendering only the largest connected component, which will be the main molecular surface (Figure 1, Right vs. Figure 3, Top).

Textured Surfaces Textured surfaces can help improve surface-shape perception [21, 22]. We therefore apply an optional Perlin noise solid texture [31] to the molecular surface. This can be especially helpful when zoomed in close to the cavity surface (Figure 11).

Ligand Visualization To understand the interaction between enzymes and ligands, it can be useful to display the ligand within the active site of the enzyme. Figure 11 shows comparison views incorporating stick models of the ligand within the surface cavity, as well as specified enzyme residues of interest. The carbon backbones are colored grey. Other representations of the ligand and residues, such as spheres or van der Waals surfaces, could also be used.

Backface Opacity Modulation When displaying the ligand within the cavity, portions of the ligand can be obscured within small pockets of the cavity. To enable visualization of the ligand in these areas, further opacity modulation can be applied to render back-facing polygons more transparent (Figure 11). The opacity of back-facing polygons is modulated as:

$$\alpha = \alpha_{in} * \left(1.0 - (\hat{N} \cdot \hat{E}) * C\right), \qquad (6)$$

where α_{in} is the opacity after Equation 4 and optionally Equation 5 are applied, $\hat{N} \cdot \hat{E}$ is the dot product of the surface normal and the eye vector, and C controls the overall opacity reduction. This equation selectively reduces the opacity of back-facing surfaces more

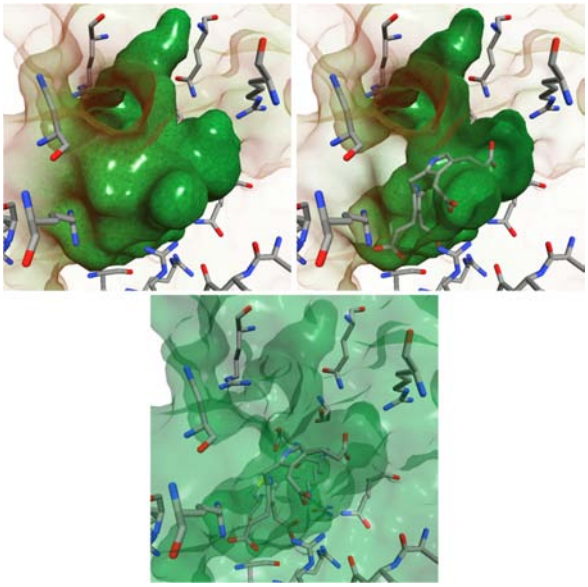


Figure 11: The top views show AOOM renderings of a closeup of the inner cavity. The image on the left shows the back-facing surface with full opacity, and the image on the right shows the back-facing surface rendered with a view-dependent opacity to reveal the ligand within. The bottom image shows the same viewpoint using standard transparency.

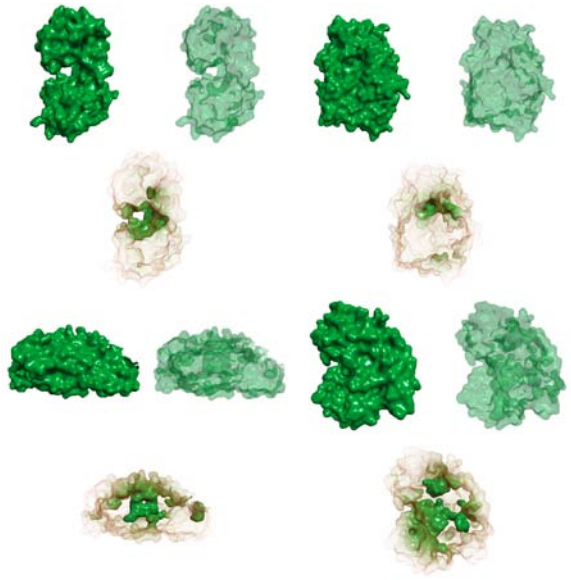


Figure 12: Various enzymes rendered using AOOM (bottom) to enable better perception of inner structure than with standard transparency (top right).

where the surface faces the viewer, providing subtle edge highlighting of the back-facing surface. For future work, it may be interesting to apply texture-based transparency methods to back-facing polygons to enable improved perception of the outer and inner surfaces of these pockets [24, 33, 44].

5 CONCLUSION AND FUTURE WORK

We have presented ambient occlusion opacity mapping (AOOM), a novel technique for viewing inner molecular surface structure. AOOM uses ambient occlusion information, a measure of the "hiddenness" of a particular point on a surface, to render occluded areas of a surface more opaque than non-occluded areas. We have shown the application of AOOM to the specific problem of visualizing the active sites of enzymes, as our collaborators have successfully used AOOM to better understand the structure of this inner cavity. Smoothing the ambient occlusion information over the surface enables control over the scale of the cavities to highlight. Color and opacity controls, including silhouette-edge highlighting, have also proven useful in highlighting the inner cavity of interest. AOOM is more effective than techniques such as transparency, clipping planes, and focal regions, and works for cases where cavity extraction techniques such as CASTp fail.

We have implemented AOOM via extensions to the Visualization Toolkit (VTK), and have created a test application using this code. Future work includes incorporating AOOM into existing molecular visualization packages, such as PyMOL, to take advantage of features, including measurements and ribbon-style rendering, that our collaborators already find useful.

It may also prove useful to apply AOOM in fields other than molecular visualization. Specifically, medical visualization and oil and gas visualization could benefit from AOOM, as both fields often work with data sets exhibiting inner structures of interest that may be occluded.

6 ACKNOWLEDGMENTS

We would like to thank Charles Lewis and Kenneth Ashe II from the Department of Biochemistry and Biophysics at the University of North Carolina at Chapel Hill for their help with this work.

REFERENCES

[1] Jean-Daniel Boissonnat, Olivier Devillers, and Jacqueline Duquesne. Computing Connolly surfaces. *J. Molecular Graphics*, 12(1):61–62, 1994.

[2] David Borland. *Flexible Occlusion Rendering for Improved Views of Three-Dimensional Medical Images*. PhD thesis, Department of Computer Science, University of North Carolina at Chapel Hill, 2007.

[3] David Borland, John P. Clarke, Julia R. Fielding, and Russell M. Taylor II. Volumetric depth peeling for medical image display. In Robert F. Erbacher, Jonathan C. Roberts, Matti T. Gröhn, and Katy Börner, editors, *Visualization and Data Analysis, Proceedings of SPIE-IS&T Electronic Imaging 2006*, volume 6060, pages 35–45, January 2006.

[4] Rob Bredow. Renderman on film: Combining CG and live action, Course 16: RenderMan in Production. In *ACM SIGGRAPH 2002 Course Notes*, 2002.

[5] Stefan Bruckner, Sören Grimm, Armin Kanitsar, and M. Eduard Gröller. Illustrative context-preserving exploration of volume data. *IEEE Transactions on Visualization and Computer Graphics*, 12(6):1559–1569, 2006.

[6] Ho-Lun Cheng and Xinwei Shi. Quality mesh generation for molecular skin surfaces using restricted union of balls. In *Proceedings of IEEE Visualization*, pages 399 – 405, 2005.

[7] Gregory Cipriano and Michael Gleicher. Molecular surface abstraction. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1608–1615, 2007.

[8] Michael L. Connolly. Solvent-accessible surfaces of proteins and nucleic acids. *Science*, 221(4612):709–713, 1983.

[9] Michael L. Connolly. The molecular surface package. *J. Molecular Graphics*, 11(2):139–141, 1993.

[10] Michael L. Connolly. Molecular surfaces: A review. *Network Science*, online article `http://www.netsci.org/Science/Compchem/feature14.htm`, 1996.

[11] Carlos Correa and Kwan-Liu Ma. The occlusion spectrum for volume classification and visualization. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1465–1472, 2009.

[12] Doug DeCarlo, Adam Finkelstein, and Szymon Rusinkiewicz. Interactive rendering of suggestive contours with temporal coherence. In *Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering*, pages 15–24, 2004.

[13] Doug DeCarlo, Adam Finkelstein, Szymon Rusinkiewicz, and Anthony Santella. Suggestive contours for conveying shape. In *Proceedings of SIGGRAPH*, pages 848–855, 2003.

[14] Paul J. Diefenbach. *Pipeline rendering: Interaction and realism through hardware-based multi-pass rendering*. PhD thesis, Department of Computer Science, University of Pennsylvania, 1996.

[15] Joachim Diepstraten, Daniel Weiskopf, and Thomas Ertl. Transparency in interactive technical illustration. *Computer Graphics Forum*, 21(3):317–325, 2002.

[16] Joachim Diepstraten, Daniel Weiskopf, and Thomas Ertl. Interactive cutaway illustrations. *Computer Graphics Forum*, 22(3):523–532, 2003.

[17] Joe Dundas, Zheng Ouyang, Jeffery Tseng, Andrew Binkowski, Yaron Turpaz, and Jie Liang. CASTp: Computed atlas of surface topography of proteins with structural and topographical mapping of functionally annotated residues. *Nucleic Acids Research*, 34:W116–W118, 2006.

[18] Cass Everitt. Interactive order-independent transparency. Technical report, Nvidia Corporation, 2002.

[19] Jan Fischer, Dirk Bartz, and Wolfgang Strasser. Illustrative display of hidden iso-surface structures. In *Proceedings of IEEE Visualization*, pages 663–670, 2005.

[20] Athanasios Gaitatzes, Yiorgos Chrysanthou, and Georgios Papaioannou. Presampled visibility for ambient occlusion. *Journal of WSCG*, 16(1-3):17–24, 2008.

[21] James J. Gibson. *The Perception of the Visual World*. Houghton Mifflin, 1950.

[22] James J. Gibson. *The Ecological Approach to Visual Perception*. Houghton Mifflin, 1979.

[23] David S. Goodsell. Visual methods from atoms to cells. *Structure*, 13(3):347–354, 2005.

[24] Victoria Interrante, Henry Fuchs, and Stephen M. Pizer. Conveying the 3D shape of smoothly curving transparent surfaces via texture. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):98–117, 1997.

[25] Adam G. Kirk and Okan Arikan. Real-time ambient occlusion for dynamic character skins. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, April 2007.

[26] Hayden Landis. Production-ready global illumination, Course 16: RenderMan in Production. In *ACM SIGGRAPH 2002 Course Notes*, 2002.

[27] Chang Ha Lee and Amitabh Varshney. Computing and displaying intermolecular negative volume for docking. In G. M. Nielson G.-P. Bonneau, T. Ertl, editor, *Scientific Visualization: The Visual Extraction of Knowledge from Data*. Springer-Verlag, ISBN 3-540-26066-8, 2005.

[28] G. V. Louie, P. D. Brownlie, R. Lambert, J. B. Cooper, T. L. Blundell, S. P. Wood, M. J. Warren, S. C. Woodcock, and P. M. Jordan. PDB #: 1pda: Structure of porphobilinogen deaminase reveals a flexible multidomin polymerase with a single catalytic site. *Nature*, 359:33–39, 1992.

[29] Rakesh Mullicka, R. Nick Bryanb, and John Butmana. Confocal volume rendering: Fast segmentation-free visualization of internal structures. In Seong K. Mun, editor, *Medical Imaging 2000: Image Display and Visualization*. Proceedings of SPIE, SPIE Vol. 3976, 2000.

[30] Arthur J. Olson and Michael E. Pique. Visualizing the future of molecular graphics. *SAR and QSAR in Environmental Research*, 8(3-4):233–247, 1998.

[31] Ken Perlin. Improving noise. *Computer Graphics*, 35(3), 2002.

[32] Christof Rezk-Salama and Andreas Kolb. Opacity peeling for direct volume rendering. *Computer Graphics Forum*, 25(3):597–606, 2006.

[33] Penny Rheingans. Opacity-modulating triangular textures for irregular surfaces. In *Proceedings of IEEE Visualization*, pages 219–225, 1996.

[34] Frederic M. Richards. Areas, volumes, packing and protein structure. *Annual Review of Biophysics and Bioengineering*, 6:151–176, 1977.

[35] E. H. Rydberg, C. Li, R. Maurus, C. M. Overall, G. D. Brayer, and S. G. Withers. PDB #: 1kbb: Mechanistic analyses of catalysis in human pancreatic alpha-amylase: detailed kinetic and structural studies of mutants of three conserved carboxylic acids. *Biochemistry*, 41(13):4492–4502, April 2002.

[36] Michel F. Sanner, Arthur J. Olson, and Jean-Claude Spehner. Fast and robust computation of molecular surfaces. In *Proceedings of the eleventh annual symposium on Computational geometry*, pages 406 – 407, 1995.

[37] Perumaal Shanmugam and Okan Arikan. Hardware accelerated ambient occlusion techniques on GPUs. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 73–80, April 2007.

[38] Peter-Pike Sloan, Naga K. Govindaraju, Derek Nowrouzezahrai, and John Snyder. Image-based proxy accumulation for real-time soft global illumination. In *Proceedings of the 15th Pacific Conference on Computer Graphics and Applications*, pages 97–105, 2007.

[39] Marco Tarini, Paolo Cignoni, and Claudio Montani. Ambient occlusion and edge cueing to enhance real time molecular visualization. In *Proceedings of IEEE Visualization*, 2006.

[40] John Tate. Molecular visualization. In Philip E. Bourne and Helge Weissig, editors, *Structural Bioinformatics*, chapter 23. Wiley-Liss, 2003.

[41] Amitabh Varshney and Jr. Frederick P. Brooks. Fast analytical computation of richardï¿½s smooth molecular surface. In *Proceedings of IEEE Visualization*, pages 300–307, 1993.

[42] Ivan Viola, Armin Kanitsar, and Meister Eduard Gröller. Importance-driven volume rendering. In *Proceedings of IEEE Visualization*, pages 139–146, 2004.

[43] Colin Ware. *Information Visualization: Perception for Design*. Morgan Kaufmann Publishers, second edition, 2004.

[44] Chris Weigle and Russell M. Taylor II. Visualizing intersecting surfaces with nested-surface techniques. In *Proceedings of IEEE Visualization*, pages 503–510, 2005.

Interactive Volume Rendering Aurora on the GPU

Orion Sky Lawlor* Jon Genetti†
Department of Computer Science, University of Alaska Fairbanks

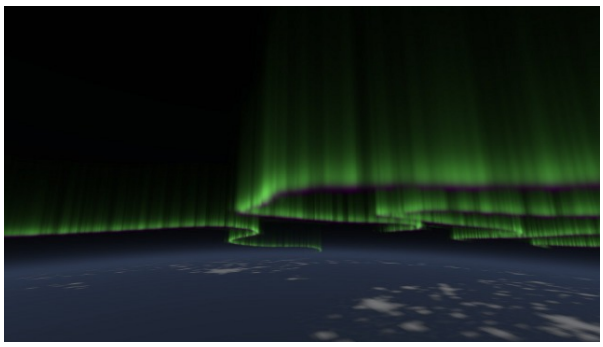


Figure 1: Our rendered aurora, 60km above Finland.

ABSTRACT

We present a combination of techniques to render the aurora borealis in real time on a modern graphics processing unit (GPU). Unlike the general 3D volume rendering problem, an auroral display is emissive and can be factored into a height-dependent energy deposition function, and a 2D electron flux map. We also present a GPU-friendly atmosphere model, which includes an integrable analytic approximation of the atmosphere's density along a ray. Together, these techniques enable a modern consumer graphics card to realistically render the aurora at 20–80fps, from any point of view either inside or outside the atmosphere.

Keywords: Volume rendering, aurora borealis, atmospheric scattering.

1 THE AURORA

The aurora borealis and aurora australis are beautiful phenomena that have fascinated viewers in Earth's polar regions for centuries. Aurora are generated when charged particles trapped by a planet's magnetic field collide with and excite gas in the upper atmosphere.

*e-mail:lawlor@alaska.edu
†e-mail:jdgenetti@alaska.edu

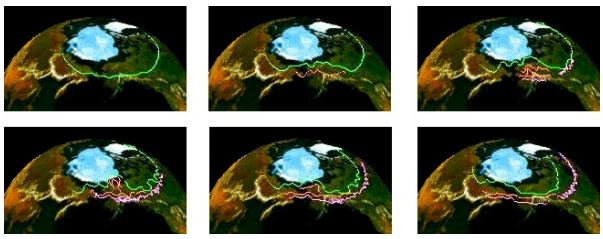


Figure 2: Global progress of a typical auroral substorm.

On Earth, these charged particles rarely penetrate below 50 kilometers altitude, and the aurora become difficult to discern above 500 kilometers due to the thin atmosphere.

The charged particle fluxes visible as auroral displays are driven by magnetohydrodynamics that are complex and the details are poorly understood, but the effects can be qualitatively described. As is typical in magnetohydrodynamics, magnetic effects expel currents from the body of a conductive plasma, compressing the charged particle currents flowing through the magnetosphere into thin sheets around one kilometer thick. As these current sheets are bent along magnetic field lines and intersect the atmosphere, they become visible as auroral "curtains," long linear stripe-like features. Depending on the activity level of the aurora, curtains can be nearly featureless greenish blur, or an extremely complex and jagged path.

A typical "auroral substorm" [Aka64] begins with simple, smooth curtains. These then grow and begin to fold over during substorm onset, resulting in many overlapping and interacting curtains, which become more and more complex and fragmentary as the substorm breaks up, and finally substorm recovery gives dim pulsating aurora. Recent work by Nishimura et al. [Nis10] has linked ground observations of pulsating aurora to space-based observations of electromagnetic waves deep in Earth's magnetotail, using the THEMIS satellites.

Because the detailed interactions of the charged particles and magnetic fields that drive auroral substorms are poorly understood, for rendering purposes we approximate their effect. We represent an auroral curtain's path using a time-dependant 2D spline curve "footprint," which are animated by hand to match the broad global outlines of an auroral substorm as it moves over the surface of the planet as shown in Figure 2.

1.1 Algorithm Overview

In this paper, we present a combination of techniques to interactively render the aurora on modern graphics hardware. To summarize our interactive GPU rendering algorithm:

1. We begin with aurora curtain footprints, described in Section 2, stored as 2D splines curving along the planet's surface.
2. We add 2D complexity to those curtain footprints by wrapping a long thin fluid dynamics simulation along them as described in Section 2.
3. We preprocess the curtain footprints into a 2D distance field described in Section 3.2, and stored in another GPU 2D texture and used to accelerate rendering.
4. We stretch the curtains into 3D using an atmospheric electron deposition function, as described in Section 2.1. The deposition function is expensive and constant, so it is stored as a GPU texture lookup table.
5. For each frame, we shoot rays from the camera through each pixel onscreen. Any camera model may be used.
6. For each ray, we determine the portion of the ray that intersects the aurora layer and atmosphere, and determine the layer compositing order as described in Section 3.1.
7. To intersect a ray with an aurora layer, we step along the ray at conservative distances read from the distance field, as described in Section 3.2. At each 3D sample point, we sum up the auroral emission as the product of the 2D curtain footprint and the vertical deposition function.
8. To intersect a ray with the lower atmosphere, we evaluate a closed-form airmass approximation as described in Appendix A.
9. Final displayed pixels are produced by compositing together the resulting aurora, atmosphere, and planet colors followed by an sRGB gamma correction, as described in Section 3.3.

Section 4 describes the performance of our algorithm on various graphics hardware.

2 MODELING THE AURORA IN 3D

Because curtains become fragmented and complex during the highly excited periods of an auroral substorm, splines alone do not convey the complexity of real curtains, as illustrated in Figures 3 and 4. Several approaches have been used to simulate this complexity,

Figure 3: Photograph of auroral curtains during a moderate substorm. The shutter was open for four seconds.

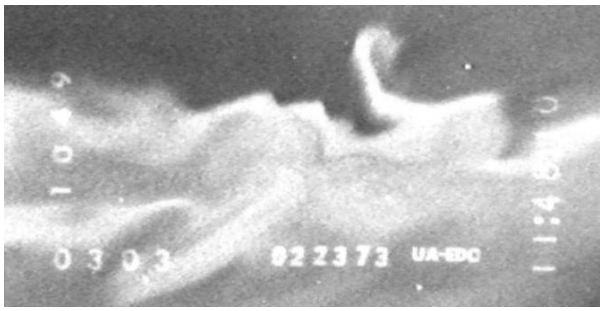


Figure 4: High-speed video of a portion of a very active curtain. Field of view is 4km wide.

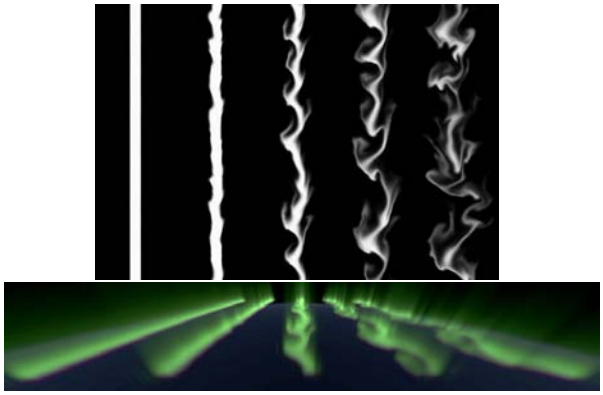


Figure 5: Portions of the 2D fluid dynamics simulations we use to model small-scale curtain complexity, and the resulting 3D auroral curtains.

such as raycasting caustics, but we find the phenomena are better matched by a fluid dynamics simulation.

To simulate aurora curtain footprint complexity, we use an simple 2D Stam-type [Sta99] fluid advection simulator. We use a multigrid divergence correction approach for the Poisson step, which is both asymptotically faster than an FFT or conjugate gradient approach, and makes the simulator amenable to a graphics hardware implementation. The simulator is solving a Kelvin-Helmholtz instability problem, with the fluid shear zone lying along the flux center of the auroral curtain, as illustrated in Figure 5. We perform the simula-

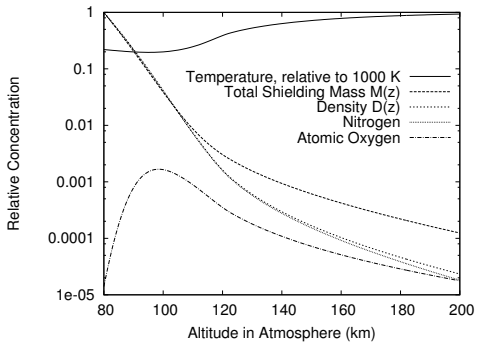


Figure 6: MSIS atmosphere as a function of altitude.

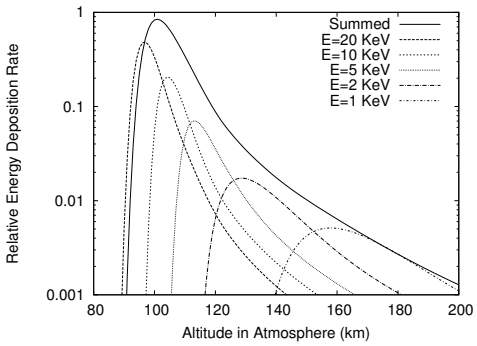


Figure 7: Auroral energy as a function of altitude.

tion in a long vertical domain with periodic boundary conditions, so we could replicate the same simulation along an arbitrarily long spline. The resulting simulated auroral curtain is stretched along the spline that defines the center line of the curtain. For quiet early periods during the substorm, we use the initial steps of the simulation, before substantial turbulence has distorted the smooth initial conditions; later more chaotic curtains are represented using later steps in the simulation, when the simulation's fluid turbulence results in a very complex electron flux pattern. The magnetosphere's actual plasma dynamics are of course very different from simple Navier-Stokes fluids, but this simulation seems to approximate the final turbulent appearance of the aurora reasonably well.

2.1 Aurora Vertical Deposition

We use splines to impose the global location of the auroral curtains, and fluid dynamics to approximate the small-scale variations in brightness, but both of these give only an electron flux footprint on the surface of the planet, in 2D. To create a full 3D volume model of the aurora, we must specify how the electrons are deposited through the atmosphere, via an electron deposition function.

The depth that charged particles penetrate the atmosphere depends on both the velocity of the charged particles and the atmosphere's state. However, the state of the upper atmosphere is not constant, due to variable energy input from solar radiation, ground-based upward travelling radiation, and even variable auroral energy deposition itself. Since the auroral energy deposition profile depends on a variety of factors, including feedback due to auroral heating, an exact deposition model would require us to simulate the spatial and temporal variations in the upper atmosphere's density, temperature, and chemistry. Software exists to do this, such as NCAR's thermospheric general circulation model, but it is not amenable to either the GPU or to realtime interactive simulation. Instead, we begin with the standard MSIS-E-90 atmosphere [Hed91], as shown in Figure 6.

We then apply the Lazarev charged particle energy deposition model [Lum92], which is still the definitive model for low-energy auroral electrons [Fan08]. The inputs to the Lazarev model are the particle energy E and the atmosphere's mass M_z and density D_z at the desired height z, and the output is the auroral energy deposition rate A_z:

E Initial energy of incoming particles, in thousands of electron-volts [keV]. For aurora, this is 1–30keV [Fan08]. These equations work well below 32keV.

z Altitude above surface to evaluate deposition.

D_z Atmosphere's density at altitude z [g/cm^3]. This is listed directly in the MSIS data.

$M_z = \int_z^\infty D_{z'} dz'$ Atmosphere's total shielding mass above altitude z [g/cm^2].

$M_E = 4.6 \times 10^{-6}\, E^{1.65}$ Characteristic shielding mass for particles of energy E [g/cm^2]

$r = M_z / M_E$ Relative penetration depth [unitless]

$L = 4.2\, r\, e^{-r^2 - r} + 0.48\, e^{-17.4\, r^{1.37}}$ Lazarev's unscaled interaction rate [unitless]

$A_z = L\, E\, (D_z / M_E)$ Aurora energy deposition rate at altitude z.

The result of the Lazarev deposition model is shown in Figure 7 for several discrete input energies, as well as a sum over energies from 1keV to 20keV. Various parameterizations of this deposition function exist, such as the popular Thermospheric General Circulation Model [Rob87], which assumes a Maxwellian distribution of electron energies. TGCM is actually simple enough to evaluate per pixel at runtime, as we explore in Section 4. However, both the Lazarev or TGCM models need an atmosphere model as input, and the thermosphere's density profile D_z is complex, as shown in Figure 6. Since we will need a lookup table to store the atmosphere's shielding mass and density, we simply pre-evaluate the deposition function for various energies and store the result in a table.

2.2 Prior Work in Aurora Modeling

We extend the excellent and rigorous aurora rendering work of Baranoski et al. [Bar00] in several ways. First, this prior work forward maps aurora curtain points onscreen followed by a guassian blur, while our renderer walks backward along camera rays accumulating visible energy. Our raytracing approach allows us to render to arbitrary resolutions and produce sharp rendered images. Second, we provide an interactive GPU implementation which includes the effect of the lower atmosphere on the aurora and allows us to render the aurora from any point inside or outside the atmosphere. In the prior work, electron-atmosphere impacts are simulated explicitly, while we simply look up their well known altitude dependent statistical energy deposition function. Finally, the prior work's curtains are constructed from a combination of sine wave with phase shift oscillations and a caustic-type electron beam deflection model; while our curtains begin as splines, with smaller turbulent deflections applied via a fluid dynamics simulation.

The later work of Baranoski et al. [Bar05] presents a detailed physically plausible model of the magnetohydrodynamics of a charge sheet's path through the magnetosphere prior to becoming visible as an auroral curtain. There appears to be an almost exact analogy between this work and our fluid dynamics simulation of curtain dynamics: electric charges with inertia interact via an electrostatic field, while fluid parcels with inertia interact via a pressure field. Both electrodynamic and fluid dynamic simulations use a multigrid Poisson solver to control field divergence, and the results appear roughly similar as well. One difference is we have not yet attempted to specialize our initial conditions to generate the spiral structures visible as auroral surges.

3 GPU RAYTRACING THE AURORA

Raytracing is a rendering technique that finds a scene's color along a ray by intersecting the ray with the scene geometry. Raytracing is computationally demanding, and the first interactive raytracers used a combination of carefully constructed scenes (such as a set of spheres) and massive parallel computing horsepower. University of Utah researchers [Par98] used a large shared-memory machine for this, while John Stone's Tachyon [Sto98] used a network of distributed-memory workstations. GPU raytracing is such a natural fit that initial work in this area [Pur02] actually preceded fully programmable GPU hardware, and an abundance of modern work exists. Similarly, volume rendering via raytracing is a venerable and well known technique [Kaj84].

3.1 Aurora rendering geometry

The aurora are almost perfectly emissive phenomena, since the degree of absorption and scattering by the atmosphere is vanishingly small around 100km altitude. Even at sea level air's optical properties are reasonably close to that of vacuum, and at 100km altitude the air's density is a millionfold smaller. The isotropic emissions, and lack of absorption and scattering, simplifies Kajiya's rendering equation [Kaj84] for the aurora layer into a single integral along the path of the ray.

Since aurora only appear in the upper layers of the atmosphere, we can treat them as a separate purely emissive "aurora layer." Below 80km is the bulk of the lower atmosphere, which both absorbs and scatters light as discussed in Appendix A. Underneath all of this is the planet's surface. Because the lower atmosphere includes scattering, implemented using alpha blending, we must composite the layers in the correct order.

A general-purpose raytracer typically uses recursion to resolve the depth order of multiple layers of translucent geometry that intersect a ray, but this general solution is not appropriate in our case. First, GPU hardware that directly supports recursion was only introduced in 2010 with the NVIDIA Fermi line, and most current cards do not directly support recursion. Second, even where it is supported this recursive search for geometry is expensive, typically requiring $O(n^2)$ intersection tests to determine the depth order of n translucent layers, and we find the many branches required can become a limiting factor in a high performance GPU raytracer.

Thus instead of a recursive search, for each ray we programmatically determine the correct compositing order of the intersected geometry, as summarized in Figure 8. The easy case is 8(a), where the ray misses all geometry and heads out into deep space. Case 8(b) is a ray that enters the aurora layer, accumulates some emitted energy, and exits. The most complex case is 8(c), where the aurora layer is entered twice: once before the atmosphere, then some aurora light is scattered out by the atmosphere, and finally a disjoint stretch of aurora layer emits more light into the ray. Finally, case 8(d) begins on the planet's surface, whose light is attenuated by the atmosphere, and then some aurora light is picked up before reaching the viewer. The same cases apply for a viewer inside the aurora layer. For a viewer inside the lower atmosphere, the only two compositing possibilities are atmosphere then planet, or atmosphere then aurora.

One limitation of our explicit ray compositing order is we do not support atmospheric refraction. However, Earth's atmosphere only very gently refracts rays, resulting in a maximum curvature near the horizon which is less than 1/6 of the planet's curvature, so we feel it is acceptable to ignore atmospheric refraction.

Given a portion of a ray that intersects the aurora layer, in principle we step through the layer accumulating aurora energy, at each step sampling the aurora curtain footprint in 2D and multiplying it by the height-dependent energy deposition function. The step size is

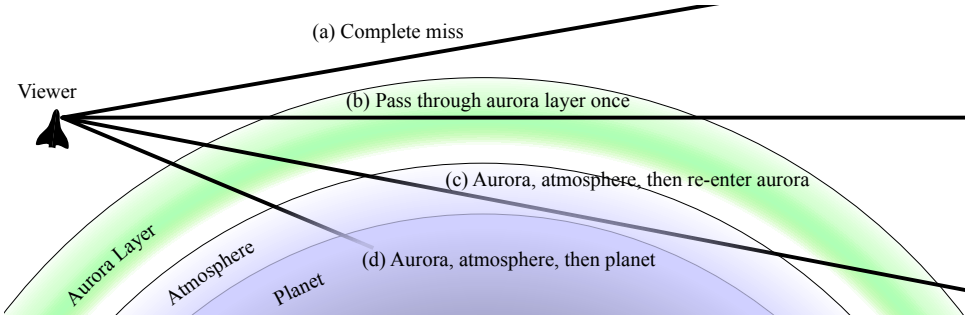


Figure 8: Possible ray/geometry intersection paths for camera rays originating outside the atmosphere.

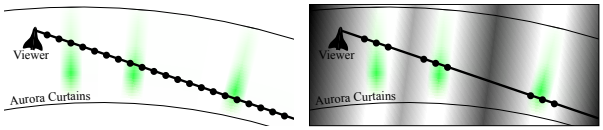


Figure 9: Naive ray stepping, left, is inefficient when curtains are sparse. Using a distance field, as shown on the right, allows the raytracer to take much larger steps in the empty spaces between curtains.

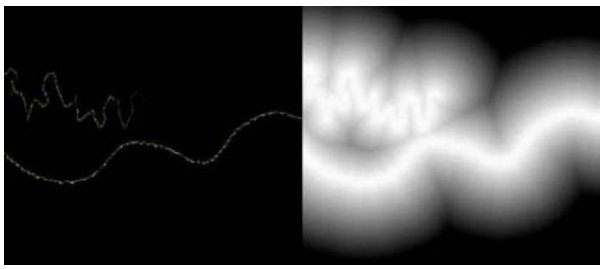


Figure 10: On the left, aurora curtain footprints. On the right, the distance field to accelerate raytracing those curtains.

a tunable parameter, with finer steps giving more aurora detail but as we show in Section 4, taking more time to compute. The step size is limited by the resolution of the aurora footprint texture: an 8192x8192 aurora footprint stretched across a 12742km diameter planet gives pixels that are 1.55km along the coordinate axes, and 2.2km diagonally. We find a 2km step size gives a reasonable quality image, but with naive sampling is quite slow. In the next section, we show how to accelerate the aurora sampling process.

3.2 Acceleration via a Distance Field

The auroral layer is hundreds of kilometers high, and wraps around a planet thousands of kilometers in diameter. Yet auroral curtains are only a few kilometers thick, so as we step along a ray we must sample the aurora layer at least every few kilometers to avoid missing curtains. Even modern GPU hardware cannot support thousands of such 3D samples per pixel in real time, since there are millions of onscreen pixels.

However, most of the auroral layer does not actually contain curtains, so if we could skip over the empty space between curtains, we could dramatically improve our overall performance. Figure 9 illustrates the problem, and the solution we use: a distance field [Coh94]. This field stores the distance to the nearest geometry, which allows the raytracer to take much larger steps through empty space.

The distance field is stored as a 2D texture, with a slightly lower resolution than the aurora curtain image. As we step along a ray, we read the step size from the distance field, so we step at a fine 2km/step rate while inside curtains; yet can take much longer steps far from curtains, up to 1000km/step, without ever skipping over a curtain. In pseudocode, our sampling loop through the aurora is as follows.

```
float t = ray.start;
while (t < ray.end) {
    vec3 P = ray.origin + ray.dir*t;
    t += distance_field(P);
    aurora += sample_aurora(P);
}
```

One surprising aspect of the GPU branch hardware is that it is actually a performance loss to skip the aurora sampling when distant from a curtain. We found it to be at least 18% slower to do the following "optimized" sampling; our other attempts at similar optimizations have been up to sevenfold slower!

```
float t = ray.start;
while (t < ray.end) {
    vec3 P = ray.origin + ray.dir*t;
    float d = distance_field(P);
    t += d;
    if (d<ε) /* inside curtain */
        aurora += sample_aurora(P);
}
```

The performance problem in this sort of loop is branch divergence, when some GPU threads take the distance-dependent branch and sample the curtain while others do not. The large GPU branch divergence penalty exceeds the savings from avoided samples, which makes

it faster to simply sample everywhere than to carefully decide whether to sample or not.

We generate the distance field from the curtain image on the GPU, but as a preprocess before rendering. We use a clever constant-time algorithm known as "jump flooding" [Ron06], which takes distance propagation steps at power of two distances to fill the distance field across the 2D image.

3.3 Coloring the Aurora

On short timescales, the upper layers of the aurora are green, while the lower layers have a purple tinge. We use the Baranoski et al. [Bar00] approach to convert the auroral emissions' isolated spectral color peaks to CIE XYZ and then a linear sRGB colorspace.

More difficult are numerical problems encountered while summing thousands of dim samples. In Baranoski et al., aurora samples are forward mapped and summed in a framebuffer, while we step along camera rays in a loop on the GPU. Because the GPU registers are floating-point, and floating-point framebuffers are expensive, a raytracer can more efficiently sum aurora samples in a high precision and high dynamic range linear colorspace. We then convert to the standard sRGB gamma of 2.2 using the following function, which outputs a color with vector magnitude equal to the old magnitude raised to the 1/2.2 power.

```
float brightness=length(color);
return color*pow(brightness,1/2.2-1);
```

4 PERFORMANCE ANALYSIS

We use the standard OpenGL Shading Language, GLSL, to implement our GPU aurora raytracer. Unlike the general-purpose GPU languages CUDA and OpenCL, the older GLSL is specialized for rendering tasks, so it directly supports graphics hardware features such as anisotropic mipmapping. Recent work on VOREEN [Men10] showed CUDA only improves performance when volume samples overlap, such as in gradient calculations. Table 1 compares the performance of our GPU aurora rendering algorithm across various GPU families, and a C++ OpenMP multicore CPU version of the algorithm. Even using four cores and nearest-neighbor texture sampling, the CPU runs about a hundred times slower than the GPU versions.

Table 2 lists the performance impact of various algorithm and parameter modifications. This is a list of alternatives not chosen for the current implementation, although many of these could still be useful.

Our raytracer acceleration distance field results in rather dramatic per-pixel performance variations, as shown in Figure 11. The corresponding frame is shown in Figure 1. Where multiple curtains cross camera rays the rendering cost can be hundreds of nanoseconds per pixel, while empty regions of space require less than

GPU	FPS
NVIDIA GeForce GTX 280	60fps
NVIDIA GeForce 8800M GTS	38fps
ATI Radeon HD 4830	23fps
Intel Q6600 2.4GHz Quad-Core CPU	0.4fps

Table 1: Comparing renderer performance across hardware. Resolution is 720p: 1280x720.

Modified Rendering Method	Cost
No distance field, use naive stepping	+350%
Make aurora layer 100km thicker	+32%
Take 1km steps through aurora, not 2km	+60%
Take 4km steps through aurora, not 2km	-33%
No table, use TGCM deposition function	+55%
No decibel map, linear deposition table	-10%
No deposition function, constant value	-14%
No curtain footprint image lookup	-14%
No exponential atmosphere	-15%
No planet texture	-0.6%
No sRGB gamma correction	-0.5%

Table 2: Performance impact of various alternatives. Positive time cost lowers framerate.

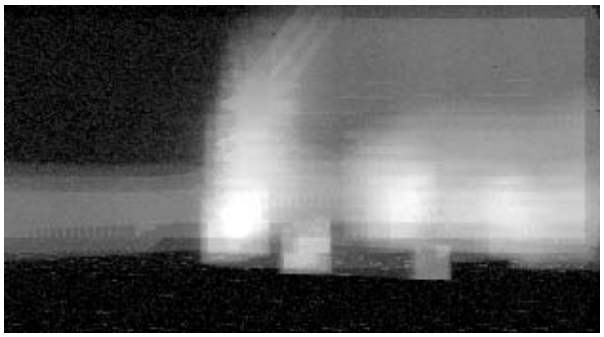


Figure 11: Measured rendering time per pixel: black represents 10ns/pixel, white represents 200ns/pixel.

ten nanoseconds per pixel. This experiment was run on the NVIDIA GeForce 8800M GTS; timings on different cards vary, but the ratios are similar. This figure is somewhat blurred due to the nature of GPU performance analysis: GPU hardware provides no means to time individual pixels, and in fact extensive GPU pipelining makes per-pixel timing difficult to even define, so instead we time overlapping blocks of 64x64 pixels. After several repetitions, the median per-block times are converted to per-pixel times by subtracting off the per-block overhead and dividing by the number of pixels. The remaining sampling jitter due to OS and driver overhead is approximately $\sigma = 2$ns/pixel.

4.1 GPU Aurora on a Powerwall

We used the parallelizing library MPIglut [Law08] to port our sequential OpenGL/GLUT aurora rendering application to a twenty-screen powerwall, as shown in

Figure 12: Interactive aurora rendering on a powerwall cluster with ten GPUs and twenty screens at 29fps.

# GPUs	Resolution	FPS	Speedup
1	1680x2100	35	1
2	3360x2100	30	1.6
4	6720x2100	27	3.0
8	6720x4200	29	6.5
10	8400x4200	29	8.2

Table 3: Parallel aurora rendering via MPIglut.

Figure 12. This was a surprisingly straightforward process, involving recompiling the rendering application with MPIglut instead of glut, and running the resulting binary. Scalability as shown in Table 3 is reasonably good, although view-dependent load imbalance becomes large when some screens must draw complex curtains and other screens only empty space; for the benchmark this impacts the two and four GPU values somewhat. The aggregate rendering rate on ten NVIDIA GeForce GTX 280 cards is a little over 29 frames per second at 8400x4200 resolution, or just over a billion finished pixels per second.

5 CONCLUSIONS

With only moderate programming effort, modern graphics hardware is capable of truly incredible amounts of computation. We have harnessed that power to render the aurora at interactive rates, but much work remains.

At the moment, our raytracer implementation stands alone, and includes no polygonal geometry. It would be relatively straightforward to extend this to a hybrid raytracer, where ordinary polygon-based geometry is first rasterized to a typical depth buffer, and these depth values are then used to limit the extent of each ray [Sch05]. This extension would allow the techniques described in this paper to add atmospheric and aurora effects to a scene that includes terrain, vegetation, spacecraft, or other geometry.

We currently render a single instantaneous snapshot of the aurora; the viewer is free to move, but the curtains are stationary. It should be straightforward to extend this to animating curtains, and we have done so offline, but image I/O and texture upload rate becomes an issue when rendering in realtime. Similarly, we currently do not integrate the curtains across the minutes-long timescale that gives high red aurora. This should be a simple change to our input curtain footprint images. Both changes should allow a detailed comparison with the widespread seconds-long-exposure photographic images of the aurora.

Since aurora are purely emissive phenomena, our atmospheric airmass model currently ignores clouds and the interesting multiple scattering effects of sunlight on the air. Incorporating these effects would allow us to simulate aurora at sunrise, or aurora rising over a thunderhead. More ambitiously, implementing a global illumination algorithm such as photon mapping or path tracing could allow aurora to cast light onto complex geometry, such as a mountainside or spacecraft.

Aurora are visible on many planets, and often display curtains and dynamics similar to those on Earth. However, the dynamics of aurora on planets without a single dominant magnetic field, such as Venus or Mars, can be quite different, and simulations would be beneficial for studying these fascinating phenomena.

ACKNOWLEDGEMENTS

The authors sincerely thank Dr. Syun-Ichi Akasofu for providing the schematics of a typical auroral substorm and the video capture in Figure 4, as well as Dr. Bill Brody for digitizing and animating that substorm as a series of continuous splines as shown in Figure 2. Our night earth texture is from NASA's Visible Earth project. Previous support for this project has been provided by the American Museum of Natural History.

REFERENCES

[Aka64] Syun-Ichi Akasofu. The development of the auroral substorm. *Planetary and Space Science*, 12(4):273 – 282, 1964.

[Bar00] G.V.G. Baranoski, J.G. Rokne, P. Shirley, T. Trondsen, and R. Bastos. Simulating the aurora borealis. In *Computer Graphics and Applications*, pages 422–432, october 2000.

[Bar05] Gladimir V. G. Baranoski, Justin Wan, Jon G. Rokne, and Ian Bell. Simulating the dynamics of auroral phenomena. *ACM Trans. Graph.*, 24(1):37–59, 2005.

[Coh94] D Cohen and Z Sheffer. Proximity clouds–an acceleration technique for 3D grid traversal. *The Visual Computer*, 11(1):27–38, 1994.

[Fan08] X. Fang, C. E. Randall, D. Lummerzheim, S. C. Solomon, M. J. Mills, D. R. Marsh, C. H. Jackman, W. Wang, and G. Lu. Electron impact

ionization: A new parameterization for 100 eV to 1 MeV electrons. *J. Geophys. Res.*, 113(A09311), 2008.

[Hed91] A. E. Hedin. Extension of the MSIS Thermosphere Model into the Middle and Lower Atmosphere. *J. Geophys. Res.*, 96(A2):1159–1172, 1991.

[Kaj84] James T. Kajiya and Brian P Von Herzen. Ray tracing volume densities. *SIGGRAPH Comput. Graph.*, 18(3):165–174, 1984.

[Law08] Orion Sky Lawlor, Matthew Page, and Jon Genetti. MPIglut: Powerwall programming made easier. *Journal of WSCG*, pages 130–137, February 2008.

[Lum92] D. Lummerzheim. Comparison of energy dissipation functions for high energy auroral electrons and ion precipitation. Technical Report UAG-R-318, Geophys. Inst., Univ. of Alaska-Fairbanks, April 1992.

[Men10] Jörg Mensmann, Timo Ropinski, and Klaus H. Hinrichs. An advanced volume raycasting technique using GPU stream processing. In *GRAPP Proceedings*, pages 190–198, 2010.

[Nis10] Y. Nishimura, J. Bortnik, W. Li, R. M. Thorne, L. R. Lyons, V. Angelopoulos, S. B. Mende, J. W. Bonnell, O. Le Contel, C. Cully, R. Ergun, and U. Auster. Identifying the driver of pulsating aurora. *Science*, pages 81–84, October 2010.

[Par98] Steven Parker, Peter Shirley, Yarden Livnat, Charles Hansen, and Peter-Pike Sloan. Interactive ray tracing for isosurface rendering. *Visualization Conference, IEEE*, 0:233, 1998.

[Pur02] Timothy J. Purcell, Ian Buck, William R. Mark, and Pat Hanrahan. Ray tracing on programmable graphics hardware. *ACM Transactions on Graphics*, 21(3):703–712, July 2002.

[Rob87] R. G. Roble and E. C. Ridley. An auroral model for the NCAR thermospheric general circulation model (TGCM). *Annales Geophysicae, Series A - Upper Atmosphere and Space Sciences*, pages 369–382, december 1987.

[Ron06] Guodong Rong and Tiow-Seng Tan. Jump flooding in GPU with applications to Voronoi diagram and distance transform. In *I3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 109–116, New York, NY, USA, 2006. ACM.

[Sch05] Henning Scharsach. Advanced GPU raycasting. In *Proceedings of CESCG*, 2005.

[Sta99] Jos Stam. Stable fluids. In *SIGGRAPH '99 Conference Proceedings*, pages 121–128, 1999.

[Sto98] John Stone. An efficient library for parallel ray tracing and animation. Master's thesis, Dept. of Computer Science, University of Missouri Rolla, 1998. http://jedi.ks.uiuc.edu/~johns/.

[You69] A.T. Young. High-resolution photometry of a thin planetary atmosphere. *Icarus*, 11(1):1–23, March 1969.

A CALCULATING AIRMASS

The integral of atmospheric density along a ray, known as "airmass," is widely used in astronomy, and we use it to approximate both the aurora light lost to the atmosphere, and night sky light added. A gravitationally bound atmosphere of uniform temperature and composition falls off in density at an exponential rate with height: $D(z) = e^{-z/H}$, with the exponential constant H known as the atmosphere's "scale height." The airmass integral along a ray parameterized by t is then:

$$A = \int_{t_s}^{t_e} D(z(t))dt = \int_{t_s}^{t_e} e^{-z(t)/H}dt$$

Even assuming a spherical planet, height varies nonlinearly along the ray path: $z(t) = \text{length}(\vec{S}+t\vec{D}) - r = \sqrt{a+bt+ct^2} - r$, so:

$$A = \int_{t_s}^{t_e} e^{-\frac{\sqrt{a+bt+ct^2}-r}{H}}dt$$

This integral cannot be solved in closed form. A trigonometric substitution [You69] allows high-order terms to be discarded, giving an integral that is easy to evaluate at the surface of the planet or at infinity, but a general raytracer requires arbitrary start and end points. We do this by approximating $z(t)/H$ with a quadratic $m + lt + kt^2$. We can eliminate the linear term l by translating the ray parameter t to t', leaving m as the height of closest approach of the ray to the planet, and k as the quadratic slope of that approach, both measured in scale height units.

$$A \approx \int_{t'_s}^{t'_e} e^{-m-kt^2}dt$$

This integral can be evaluated exactly using the error function "erf":

$$A \approx e^{-m}\sqrt{\frac{\pi}{4k}}\left(\text{erf}(\sqrt{k}t'_e) - \text{erf}(\sqrt{k}t'_s)\right)$$

Some GPU languages like GLSL do not have a built-in erf, so we use the Winitzki approximation:

$$\text{erf}(x) \approx \sqrt{1 - e^{-x^2\frac{\frac{4}{\pi}+0.147x^2}{1+0.147x^2}}}$$

Despite the plentiful transcendentals, this performs quite well on the graphics card at runtime. Despite the stacked approximations, accuracy appears quite good as well, except where numerical roundoff causes the erf difference to approach zero. This case can be handled by either falling back to a linear approximation of $z(t)$, or by interpreting the finite difference of erf values as a scaled derivative of erf: e^{-kt^2}.

Fast and Memory Efficient Feature Detection using Multiresolution Probabilistic Boosting Trees

Florian Schulze
VRVis Center for Virtual Reality and Visualization Research
fschulze@vrvis.at

David Major
VRVis Center for Virtual Reality and Visualization Research
dmajor@vrvis.at

Katja Bühler
VRVis Center for Virtual Reality and Visualization Research
buehler@vrvis.at

Abstract

This paper presents a highly optimized algorithm for fast feature detection in 3D volumes. Rapid detection of structures and landmarks in medical 3D image data is a key component for many medical applications. To obtain a fast and memory efficient classifier, we introduce probabilistic boosting trees (PBT) with partial cascading and classifier sorting. The extended PBT is integrated into a multiresolution scheme, in order to improve performance and works on block cache data structure which optimizes the memory footprint. We tested our framework on real world clinical datasets and showed that classical PBT can be significantly speeded up even in an environment with limited memory resources using the proposed optimizations.

Keywords: Feature Detection, Machine Learning, Decision Trees

1 INTRODUCTION

In the past years various methods for automatic processing and understanding of medical 3D image data have been developed. One important building block is the automatic detection of anatomical landmarks. Detection of these features stands often at the beginning of the processing pipeline: it transforms the dense volume representation into a sparse set of possible landmark locations, allowing a significant acceleration of subsequent high level segmentation methods.

However, making the transition from pure research algorithms which focus often solely on detection performance to real world radiology applications brings a number of additional requirements into consideration. The algorithm has to be able to deal with possibly limited technical resources - not all workstations in a hospital might be equipped with the newest hardware, and the algorithm shall run in the context of radiology workstation software which already occupies resources. Excellent time performance is required because automatic algorithms often substitute manual workflows while the result must be authorized and/or adjusted by the radiologist. In this case an automatic algorithm will only be used if the execution time of the algorithm is considerably shorter than the manual approach would be.

This work presents a highly optimized general purpose feature detection framework for the effective reduction of possible feature candidate positions in 3D image data as preprocessing step for more expensive object detection methods. Referring to the clinical application context, we designed our method according to the following requirements:

1. Time Performance: The result must be calculated in relatively short time (e.g. within seconds) in order to be usable in a clinical environment.

2. Memory Performance: The algorithm must also execute on standard PCs with limited technical resources.

Thus, the focus of the proposed algorithm and its implementation is on a small adaptable memory footprint while retaining as much execution speed as possible.

2 RELATED WORK

Object recognition, and local feature detection as a subdiscipline of it, are since many years core topics of computer vision research.

Point based methods beginning with the Harris corner detector [HS88] try to automatically extract points of interest from an image. Exact control of which points are extracted is not supported, therefore recognition of complex structures/areas is done by combining sets of feature points. The most prominent point detector is the SIFT algorithm [Low99] which overcomes the limitations of previous solutions by being scale, rotation and perspective invariant. However, translating SIFT, which is aimed for 2D images, to 3D volumes suffers from dramatic performance problems. Niemeijer et al. report in [NGL+09] that SIFT feature extraction on a

$200 \times 200 \times 1024$ volume downscaled by 50% takes 10 minutes to compute.

Machine learning based approaches use a (learned) classifier to decide if a specific region of an image belongs to an object. A prominent example for this class of algorithms is the method for real time face detection presented by Viola and Jones [VJ01] that uses a cascade of boosted weak classifiers. A more general approach has been proposed by Tu et al. [Tu05] by introducing Probabilistic Boosting Trees (PBT). PBTs are decision trees which use boosted learners as classifiers in each tree node. Violas boosted classifier cascades are a special case of a PBT. An alternative to PBTs is the popular d-tree forests method [MDUA07] which produces higher detection rates, but with the drawback of much higher execution costs [LK08].

PBTs have been successfully applied on tissue classification on medical images: Militzer and Vega-Higura [MV09] use PBT for bone removal in CT angiography. The volume is first split into segments using the watershed algorithm, then each segment is classified with PBT.

Fast preselection of feature candidates for more expensive high level methods is the topic of the paper of Langer and Kuhnert [LK08]. They integrate classical decision trees with simple color based features and a multiresolution scheme for candidate computation for the expensive SIFT feature detection.

The problem of the large memory footprint of volume data is often discussed in context of volume rendering. LaMar et al. [LHJ99] use an octree structure with blocks containing different resolutions, where only the needed subvolume is downloaded to graphics hardware. However, the whole volume data still has to fit into main memory. This has been improved by Guthe et al. [GWGS02] who proposed to hold the data 30:1 wavelet compressed in memory and extract needed data on demand block-wise and cache the data as long as possible.

The purpose of our feature detection method is similar to that of Langer and Kuhnert [LK08] since we also aim to reduce the list of possible candidate position as much as possible for later more expensive methods. Langer and Kuhnert tailored their algorithm especially for pre-filtering for SIFT feature computation. In difference to them we decided to use the more general PBT [Tu05]. This has several advantages: first, it is independent from SIFT features and easily adaptable to any kind of landmark/structure. Second, decision tree methods can capture large image variabilities while only need to execute $\log n$ weak classifiers. Third, they are robust against over-fitting unlike classic decision algorithms.

Our contribution. To satisfy the high performance requirements to the algorithm in a clinical environment, we extend the original PBT by integrating cascading tree nodes into normal tree building and introduce the concept of classifier sorting (Section 3.1). Both result in higher execution speed of the classifier. A second performance optimization is achieved by integrating the PBT into a multiresolution classification scheme (Section 3.2). An effective postprocessing step is introduced that applies particle filters to compute probability maps for candidate features for outlier detection (Section 3.3). The memory footprint of our feature detection framework is optimized by the introduction of a multiresolution, multi-derivative block cache data structure (Section 4). The performance of our method has been evaluated on a real world clinical usecase (Section 5).

3 ALGORITHM

In the following we explain in detail the classifier and our extensions on it (Section 3.1), the multiresolution feature detection framework (Section 3.2) and the postprocessing step based on candidate probability (Section 3.3).

3.1 Probabilistic Boosting Tree with Partial Cascading and Classifier Sorting

Probabilistic Boosting Trees. A PBT [Tu05] is a special kind of decision tree which holds at each tree node a boosted classifier. PBTs are trained top down. Based on a set of positive and negative samples a boosted classifier with a limited number of weak classifiers is trained for each tree node. On each recursion level the sample set is split using the generated classifier and the new subsets are used to train positive and negative child branches. Although multi-class classifiers are possible, we limited our implementation to the simple two-class model.

Classical Cascading. If the boosted classifier in each tree node is trained in a way that it does not produce false negative results, the resulting decision tree consists of positive child nodes only. Traversing this tree has only one sequential path and degenerates to the cascade of boosted classifiers of Viola and Jones [VJ01] (see figure 1 left). Cascading improves execution speed. It allows the classifier to early terminate and reduces

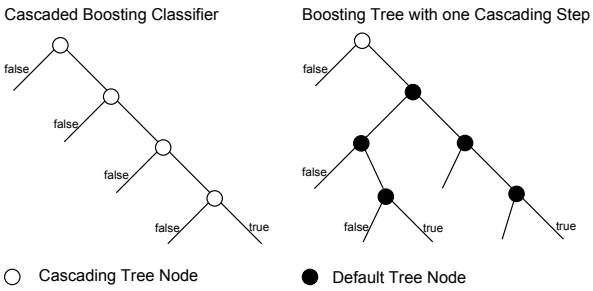


Figure 1: Probabilistic Boosting Tree. Left, tree with cascading nodes only. Right, one cascading node at the tree root followed by a default PBT.

in this way the number of classification tests, but it reduces also the flexibility of the original PBT to capture a high variability of features.

PBT with Partial Cascading. We observed that a high number of samples can be classified as false by executing only one boosted classifier (see section 5.2). This allows to combine the speed-up of cascading with the flexibilty of the PBT by placing one cascaded classifier in front of the PBT: Our tree model contains one cascading node at the root level. A negative outcome stops the classification immediately, a positive outcome is further processed using the full PBT (figure 1 right).

Classifier Sorting. We also observed that a high amount of samples can be early terminated with a cheap and fast performing classifier (see section 5.2) and that it is advantageous to use expensive classifiers only in places which are executed less often. In our model the most visited place is the cascading node at the root of the tree which can discard a large amount of samples as false. The rest of the tree is visited less frequently. Hence, we sort the expensive classifiers into the later tree nodes while the first node can only use fast executing classifiers.

Image Features. The classifier decides on a per voxel basis if the current voxel belongs to the searched structure or not. Since PBT is a so called ensemble classifier, basically every possible classification method can be integrated. However, the selection of image features has influence on detection performance and execution speed.

In the current work we integrated classifiers which make decisions based on five different image features.

1. Haar-like features with different patterns and sizes.
2. Image intensity
3. Gradients and principal curvatures
4. Region histograms based on image intensity and derivatives with different sampling resolutions and sizes.
5. Structure tensors

Haar-like features and image intensities are the features with the lowest computational costs and are therefore used for building the cascaded root. Gradients need to be computed by filtering as well as principal curvatures which need an additional Hessian analysis step. Region histogram classification multiplies the cost by the number of samples. Structure Tensors require the convolution of the gradient image with a Gaussian kernel and subsequent eigenanalysis of the structure tensor matrix. These three types of classifiers are exclusively used for the non-cascaded part of our PBT.

The chosen weak classifiers are scale variant which is adequate for our application scenario because we expect anatomical structures to have a specific size (small variations in size should be accepted anyway, larger variations because of age or gender can be covered with different detectors and pre-classification based on patient background data).

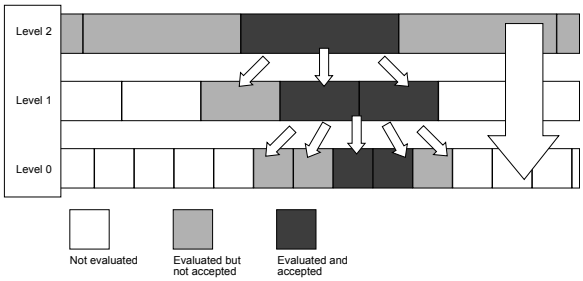


Figure 2: Multiresolution Algorithm

3.2 Multiresolution Feature Detection

The PBT with Partial Cascading is embedded into a multiresolution scheme based on a power of two Gaussian image pyramid [AAB+84] to further reduce the number of voxels to be processed.

A separate classifier C_i is trained for each resolution level. Multiresolution classification starts at the lowest resolution level n by applying classificator C_n on image I_n. Classification results in a set positively marked voxels $(p_0^{+,n},...,p_m^{+,n})$. These voxels are propagated into the next higher resolution level $n-1$ where each positive lower resolution voxel marks the voxels within the corresponding filter kernel in level $n-1$ as candidates. Classification of the current level is only computed on the remaining candidate voxel. The propagation is repeated until the original resolution (level 0) is reached. Figure 2 depicts the algorithm with a 1D example. Note that most of the high resolution voxels do not need to be checked using this scheme.

In the case of overlapping kernels some higher resolution voxels have two or more parent voxels and it can happen that a voxel is marked as positive and negative. In this case the positive mark is kept. This leads to a slight over-segmentation, but on the other hand the effect of false negative samples might be reduced, which is a wanted effect.

3.3 Filtering of Results Using Fast Probability Computation

The direct result of our feature detection algorithm is a bit mask of candidates which still might contain false positives. One method to reduce the number of false positives is to assign a probability to each candidate that reflects the confidence in its classification. The resulting probability map can then be further processed by thresholding which effectively removes outliers and/or non-maximum-suppression which only leaves the candidates which are at the center of the expected shape.

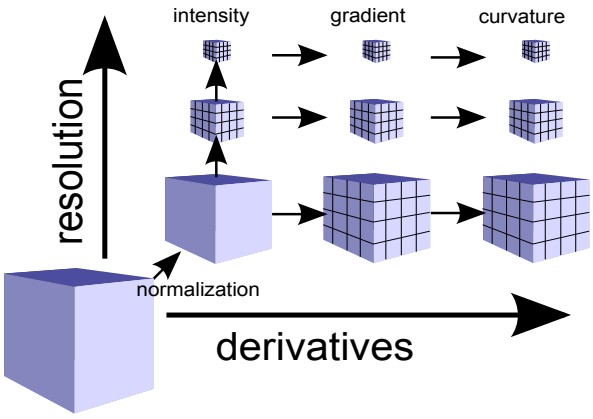


Figure 3: Datastructure: Only the base intensity volume is kept completely in memory. All other data, derivative and lower resolution volumes are computed block wise on demand.

Probabilistic boosting trees can deliver such a probabilistic classification. Drawback of this straight forward approach is the low time performance.

We observed that the result of feature detection form clusters at the feature location resembling already the searched structure (e.g. the intervertebral discs in figure 9). Thus, we propose to assign probabilities to candidates by comparing the shape of its surrounding cluster with the searched shape.

A fast option to compute this are shape particle filters which are applied in our framework. The likelihood that a candidate belongs to the searched structure is computed by applying a shape approximating the structure of interest around each candidate and by measuring the ratio of overlap of neighborhood and shape.

4 IMPLEMENTATION

4.1 Data Preprocessing

The spatial resolution of medical 3D images in a clinical environment is generally highly anisotropic. Especially the slice distances show high variability from modality to modality, from scanner to scanner depending on the used imaging protocol. The scale variant nature of the image features described in section 3.1 requires the same spatial resolution of all images to be processed.

Thus, training data as well as unseen data is preprocessed by resampling the original volume data to an isotropic voxel size that is selected based on the targeted anatomical landmark. The current implementation uses bilinear interpolation for resampling.

The resampled volume (in the following denoted as "base volume") is the basis for all following computations and the original data can be discarded at this point.

4.2 Data Management and Derivative Computation

The data management component is responsible for efficiently providing the necessary data to compute the requested weak classifiers on all resolution levels while keeping the memory footprint small and flexible.

The supported weak classifiers require intensity, gradient and principal curvature data for all positively marked voxel positions on the different levels of resolution. It is obvious that the performance of the weak classifiers decides on the performance of the whole PBT.

It is well known that filtering volume data with separated filters for derivative computation is much faster than applying a three dimensional filters per voxel individually. We currently use a $3 \times 3 \times 3$ Sobel for gradient computation, which can be replaced by any other appropriate separable filter. However, applying a separable filter for derivative computation requires to keep the whole filtered volume in memory, which might be problematic having our initial requirements in mind.

To overcome this limitation and to make the memory footprint manageable also in an environment with limited resources, we introduce a cached block structure (see Figure 3). The intensity base volume is entirely located in memory. Lower resolution volumes, gradients, structure tensors and principal curvature are organized into smaller blocks that are only computed on request. After computation, block data remains cached in memory. If the memory for allocation of new blocks gets low, the cache is partially cleaned by removing data which was accessed the longest time ago.

For fast computation of Haar-like features an additional data structure, an integral volume, is needed. This data is currently computed as a whole and kept in memory. This is due to the more complicated generation method of this data which makes it hard to compute the value block-wise on demand.

4.3 Optimized Classifier Execution

Generally each voxel can be classified individually by executing the whole boosting tree starting from the lowest resolution. However, having in mind that one voxel in a lower resolution volume has influence on a number of voxels in the higher resolution and that the data is arranged in a cached block structure, it is worth to consider a optimal execution order.

Detection of features on the whole volume or of a sub volume follows two strategies. First, feature detection is done in resolution level order. This means that the PBT for one level is executed on the whole region of interest and then all positive classified voxels are propagated to the next higher level.

Second, all per level classification is performed block wise. In this way only a small number of data blocks

must be in cache. Any other execution order (for example line wise) would cause a lot of cache misses and would likely lead to often re-computation of block data. If multiple classifiers must be applied on the same volume all classifiers are executed on each block sequentially. After the first classifier is executed the block cache remains in (partially) filled state. Data which is already cached must not be computed if the next classifier tries to access this data. Parallelization is implemented using a worker thread-pool. Classification of one block is fed into a job queue which distributes the work to the worker threads.

5 EXPERIMENTS

Our multiresolution PBT framework was tested in a real world scenario as preprocessing part for a semi-automatic annotation algorithm for the vertebral column. The task was to preselect appropriate candidates for the location of the intervertebral discs and the spinal canal.

For the intervertebral discs, three different detectors were trained to cover the different appearance of lumbar, thoracic, and cervical disks. The spinal canal could be detected by using only one detector.

5.1 Setup and Training

The algorithm has been trained and evaluated on 19 CT datasets (13 for training 6 for evaluation only) containing different parts of the vertebral column. The datasets have up to 1112 axial slices with a slice resolution of 512×512 and a slice distance between 0.62 *mm* and 3.0 *mm*. Some of the data contains pathologies (broken vertebrae, collapsed disc, scoliotic spines) as well as one cervical dataset from a child.

Experiments have shown that the thinnest intervertebral discs in the cervical section can still be distinguished if the slice distance is at least 1.5 *mm*. We therefore fixed the base volume voxel scale for this experiment as 1.5 *mm* isotropic and the datasets were resampled accordingly.

In all datasets position and location of the intervertebral discs and the spinal column have been manually labeled. Based on the given annotation, positive samples have been generated randomly inside the intervertebral disc and the spinal column. Negative samples have been generated randomly all over the volume with the constraint to have a minimal distance to positive samples of 10 *mm*.

5.2 Performance Evaluation

Time performance of the algorithm has been assessed based on a set of eleven CT volumes (six evaluation and five training datasets). The properties of the data, its original and normalized size is listed in table 1. The classifier is trained using one cascading step and allow only intensity and Haar-like features in the cascade

Volume	original size	normalized size
1	$512 \times 512 \times 202$	$106 \times 106 \times 134$
2	$512 \times 512 \times 163$	$113 \times 113 \times 108$
3	$512 \times 512 \times 361$	$144 \times 144 \times 168$
4	$512 \times 512 \times 222$	$89 \times 89 \times 148$
5	$512 \times 512 \times 249$	$170 \times 170 \times 166$
6	$512 \times 512 \times 152$	$91 \times 91 \times 101$
7	$512 \times 512 \times 277$	$245 \times 245 \times 184$
8	$512 \times 512 \times 260$	$244 \times 244 \times 179$
9	$512 \times 512 \times 1112$	$274 \times 274 \times 370$
10	$512 \times 512 \times 228$	$176 \times 176 \times 228$
11	$512 \times 512 \times 945$	$244 \times 244 \times 630$

Table 1: Properties of volumes for performance evaluation.

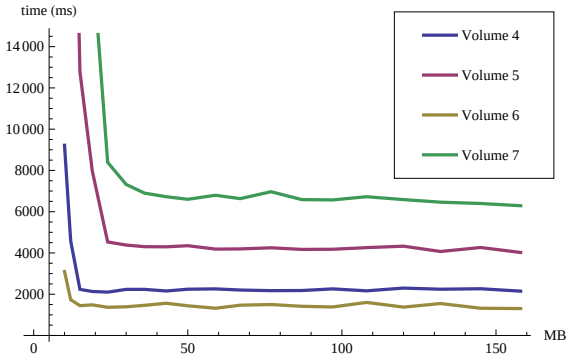


Figure 4: Memory Limits

node. Influence of the different optimizations is measured against this default. Detection performance was measured based on 8 datasets containing the 6 evaluation datasets.

Limited Memory The data structure is designed to cope with limited resources. However, reaching the bounds of memory provokes clearance of cache blocks that might have to be recomputed at a later stage of the algorithm. Figure 4 illustrates the time performance over different cache memory bounds for datasets $4-7$ and show a clear threshold ($\sim$ 25 MB) for all four datasets where the performance/memory ratio changes dramatically. This memory limit is slightly different for each dataset and depends on the dataset size. If the available memory falls below that threshold computation time rises heavily whereas performance remains stable if enough memory is available. The threshold marks the point where data blocks need to be frequently recomputed. As long as enough memory is available deletion of block data from the cache and occasional recomputation has almost no influence on performance.

Multithreading. We tested the multithreading performance of our algorithm on an Intel quad core CPU with 2.4 Ghz and hyperthreading. Figure 5 plots the computation speed over the number of threads again on datasets $4-7$. Time drops until 4 threads are used. For more threads no significant speed-up (but also no significant slowdown) can be monitored.

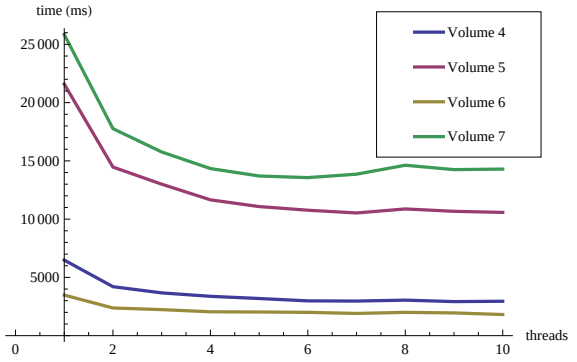


Figure 5: Plot computation time against number of threads. Tested on a quad-core with Hyperthreading.

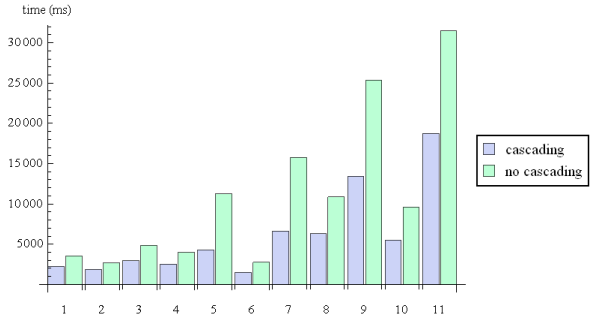


Figure 6: Detection speed comparison between PBT with (blue) and without (green) cascading on eleven different datasets.

The scaling with the number of threads below four is not linear. This is caused by the current locking strategy that prohibits accessing one block if it is currently computed by another thread. This situation mainly occurs if 2nd derivatives have to be computed that require accessing also neighboring first derivative blocks. If another thread is classifying one of these neighboring blocks at the same time it has to wait until the lock is released. This kind of collision happens more frequently as more threads are used. We expect therefore a logarithmic scaling of time performance with the number of cores as long as the locking behavior is not improved.

Cascading Speed-up. The impact of cascading on detection speed has been measured by comparing the time performance of our default detectors with detectors which are trained without including a cascading step. The result is plotted in figure 6. Over eleven datasets we measured a speed-up of $1.45 - 2.67$ for detectors including a cascading step.

Classifier Sorting Speed-up. The impact of classifier sorting is plotted in figure 7. We compare the time performance of our default detector including cascading and sorting with detectors which are allowed to use all classifiers in the cascading node. Classifier sorting results in a speed-up up to 1.65 for detectors which use sorting.

Multiresolution Speed-up. To measure the impact of multiresolution feature detection we compared de-

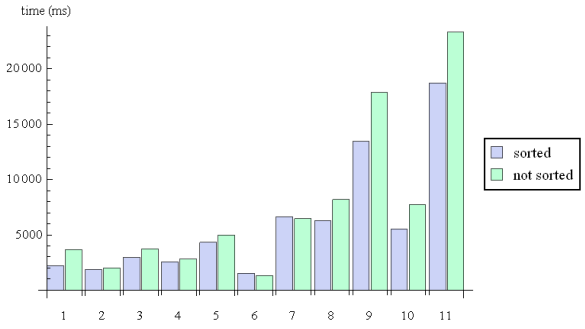


Figure 7: Detection speed comparison between PBT with (blue) and without (green) classifier sorting on eleven different datasets.

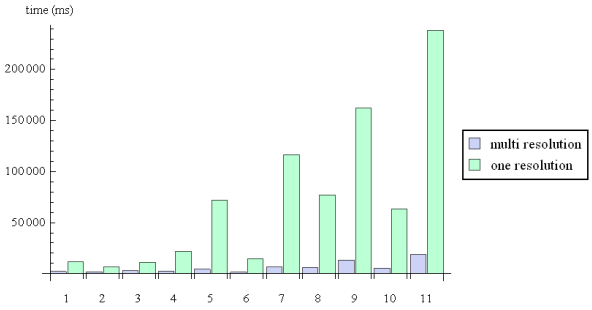


Figure 8: Detection speed comparison between multiresolution vs. one resolution.

tectors using three levels of resolution against detectors using only one level. The results are plotted in figure 8. The measured speed-up ranges between 3.44 and 17.51.

Detection Performance. Two feature detection results are depicted in Figure 9. The first row shows the detection of intervertebral discs in the lumbar section of the spine, the second row the detection of the spinal canal on a whole spine. The detection progress from lowest to highest resolution level is depicted from left to right.

The images illustrate well the effectiveness of the multiresolution scheme since already at the lowest resolution level the major part of the volume is excluded from higher resolution analysis.

The selected voxels (blue) reproduce the shape of the searched anatomical parts to a large extend. However outliers can be observed, for example inside the vertebral body (first row) or at the ventral side of the ribcage (second row). Moreover missing features can be observed as well (first row, ventral side of the topmost disc).

This observation is also reflected in recall and 1-precision plots (figure 10). Recall denotes the ratio between selected voxels within the ground truth and all possible ground truth voxels. 1-precision stays for selected voxels outside the ground truth divided by all the voxels which were selected by the feature detection (also the falsely selected ones). The evaluated data involves healthy spines (1, 2, 3, 6, 7 in figure 10) and

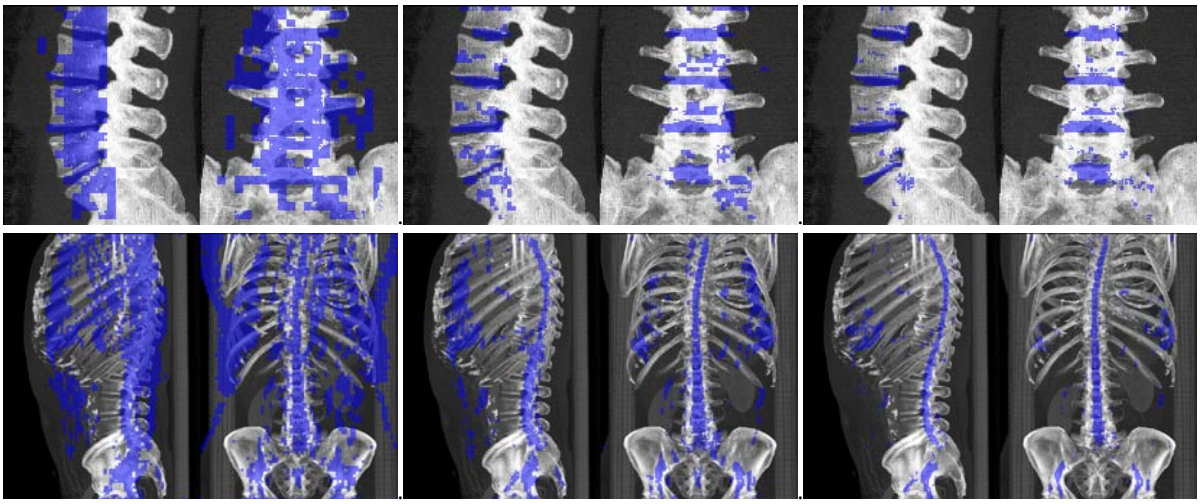


Figure 9: Coronal and sagittal images of detection results for the intervertebral disc (first row) and the spinal column (second row). Three levels of resolution document the detection process, lowest resolution left to highest resolution right.

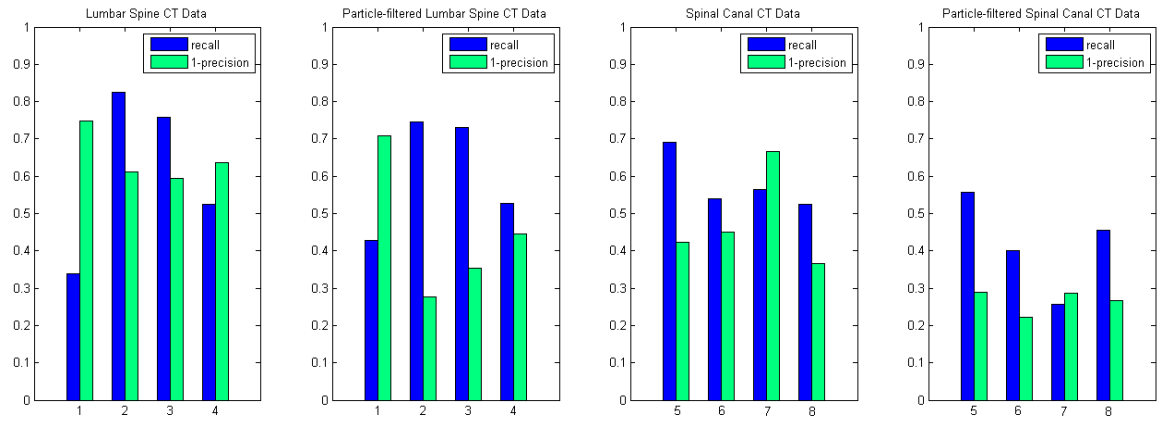


Figure 10: Recall and 1-Precision Plots

spines with diseases like scoliosis and broken vertebrae (4, 5, 8 in figure 10).

The first and the third graph show results after feature detection without any postprocessing where the high recall rates give information about good detection results of structures of interest (discs and spinal canal). However, besides the high recall rates there are also high rates of 1-precisions because of the occurrence of outliers (i.e. spongy bone within vertebrae with similar features to discs). The high 1-precision rates can be reduced by postprocessing steps such as particle filters which are visible in the second and fourth graph of figure 10. The recall rates remain fairly the same, minor reductions are due to moving towards the center voxels of the discs by particle filtering.

An example for postprocessing of the resulting feature mask is depicted in figure 11. First probabilities are computed by applying a box shape particle filter with the dimensions $9 \times 9 \times 60mm^3$. The box approximates elongated shape of the spinal canal. Second, the feature

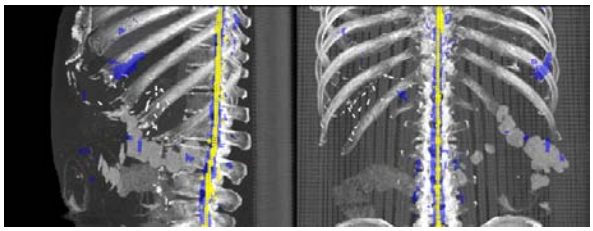


Figure 11: Feature mask (blue) post processed with particle filtering, non-maximum suppression and thresholding (yellow).

points are reduced by non-maximum suppression of the probabilities. Third, outliers are removed by thresholding the probability. The threshold is defined at $t = 0.15$.

6 DISCUSSION AND CONCLUSION

We have presented a method for time and memory efficient feature detection on medical 3D volume data. The goals and requirements formulated at the end of

Section 1 have been reached by selecting a classification based approach based on a Probabilistic Boosting Tree classifier. The classification method was improved by combining the decision tree with one cascading step and the introduction of classifier sorting. This classifier was embedded into a multiresolution framework. We could show that all optimizations together result in a huge time performance gain with an approximated speed-up factor of 20.

Multithread performance was measured to scale non linear (almost logarithmic) which is due to internal data locking. The speed-up is for state of the art quad core CPUs still significant. But to benefit from more parallelism, improvements have to be done in this section. However it is likely that more sophisticated access patterns and locking schemes can help to overcome this problem.

The behavior of the block cache data structure was evaluated in section 5.2. It is noticeable that even larger datasets require only $\sim 25MB$ for the block cache to run almost unhindered. However even under circumstances where less memory is available the algorithm will just perform slower.

Detection rate of this feature detector is not as good as it could be. We believe that other image features and filtering techniques, a finer bases scale and also a different kind of classifier could result in better detection performance. However, trading detection performance against execution speed was a conscious design decision. The results are good enough to use this method to reduce the search space for more specialized and more expensive image processing methods.

REFERENCES

[AAB+84] E.H. Adelson, C.H. Anderson, J.R. Bergen, P.J. Burt, and J.M. Ogden. Pyramid methods in image processing. *RCA engineer*, 29(6):33–41, 1984.

[GWGS02] S. Guthe, M. Wand, J. Gonser, and W. Straßer. Interactive rendering of large volume data sets. In *Visualization, 2002. VIS 2002. IEEE*, pages 53–60. IEEE, 2002.

[HS88] C. Harris and M. Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, page 50. Manchester, UK, 1988.

[LHJ99] E. LaMar, B. Hamann, and K.I. Joy. Multiresolution techniques for interactive texture-based volume visualization. In *Proceedings of the conference on Visualization'99: celebrating ten years*, pages 355–361. IEEE Computer Society Press, 1999.

[LK08] M. Langer and K.-D. Kuhnert. A new hierarchical approach in robust real-time image feature detection and matching. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pages 1 –4, dec. 2008.

[Low99] D.G. Lowe. Object recognition from local scale-invariant features. In *iccv*, page 1150. Published by the IEEE Computer Society, 1999.

[MDUA07] Christophe Marsala, Marcin Detyniecki, Nicolas Usunier, and Massih-Reza Amini. High-level feature detection with forests of fuzzy decision trees combined with the rankboost algorithm. Technical report, Université Pierre et Marie Curie-Paris, 2007.

[MV09] A. Militzer and F. Vega-Higuera. Probabilistic boosting trees for automatic bone removal from CT angiography images. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 7259 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, February 2009.

[NGL+09] M. Niemeijer, M.K. Garvin, K. Lee, B. van Ginneken, M.D. Abràmoff, and M. Sonka. Registration of 3D spectral OCT volumes using 3D SIFT feature point matching. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 7259, page 51, 2009.

[Tu05] Z. Tu. Probabilistic boosting-tree: learning discriminative models for classification, recognition, and clustering. In *ICCV*, volume 2, pages 1589 –1596, 2005.

[VJ01] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2001)*, volume 1, pages I–511 – I–518, 2001.