Automatic Spine Identification in Abdominal CT Slices using Image Partition Forests

Stuart Golodetz Irina Voiculescu Stephen Cameron

Oxford University Computing Laboratory
Wolfson Building, Parks Road
Oxford OX1 3QD

Abstract

The identification of key features (e.g. organs and tumours) in medical scans (CT, MRI, etc.) is a vital first step in many other image analysis applications, but it is by no means easy to identify such features automatically. Using statistical properties of image regions alone, it is not always possible to distinguish between different features with overlapping greyscale distributions. To do so, it helps to make use of additional knowledge that may have been acquired (e.g. from a medic) about a patient's anatomy. One important form this external knowledge can take is localization information: this allows a program to narrow down its search to a particular region of the image, or to decide how likely a feature candidate is to be correct (e.g. it would be worrisome were the aorta identified as running through the middle of a kidney). To make use of this information, however, it is necessary to identify a suitable frame of reference in which it can be specified. This frame should ideally be based on rigid structures, e.g. the spine and ribs. In this paper, we present a method for automatically identifying the spine in image partition forests of abdominal CT slices as a first step towards defining a robust coordinate system for localization.

1. Introduction

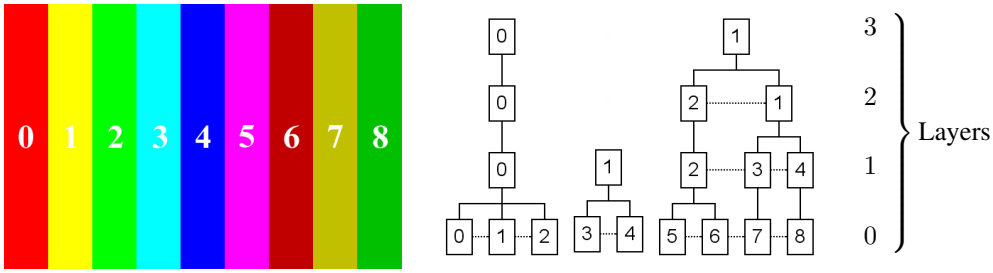
From 3D visualization [9], to volume estimation [5], to automatic landmark-based registration, there is a plethora of medical imaging applications which rely on initially knowing where key features in medical images are to be found. The process of identifying features such as organs and tumours is difficult to automate in the case of medical scans (e.g. CT and MRI), for the reasons we outlined in [3]: the boundaries between adjacent features can be indistinct, and it is difficult to encode positive shape constraints for features which may differ significantly from slice to slice. Furthermore, the greyscale (Hounsfield Unit, in the case of CT) distributions for distinct features may overlap, making the features difficult to distinguish by values alone.

Radiologists, who are expert at reading medical scans, do not rely merely on greyscale values to tell features apart, but make use of their knowledge of anatomy to decipher an image. This anatomical knowledge can take many forms, but one of the most straightforward is localization information, i.e. knowing which features they expect to see in certain places in the image. Computer programs can equally make good use of this information to narrow down their search for a feature to a particular region of the image, or to validate the candidate features suggested by other algorithms.

To incorporate localization information into a computer program, it needs to be supplied in an image-independent way, i.e. relative to a fixed frame of reference. It makes sense to search for (say) kidneys in regions specified relative to a fixed point such as the spine [4, 8]; it makes far less sense to search for them in regions specified purely in image coordinates, which have little anatomical relevance.

In this paper, we present a region flooding method for automatic spine identification in abdominal CT slices, as an incremental step towards defining a robust frame of reference for localization purposes in such images. The approach works by selecting the candidate spine in an image partition forest (**IPF**) of the image (see §2). In our previous work [3], we showed how ribs could be automatically identified in CT images using a Bayesian approach; we ultimately plan to combine a refinement of that approach with this current work to define a suitable coordinate system for localization. Existing spine segmentation methods [1, 10] achieve good results, but require significant work to implement: for localization purposes, they are unnecessarily complex, since we are only interested in using the spine to establish a coordinate system. By contrast, our method is well-suited to the specific application of localization because it produces good results whilst remaining simple to understand and easy to implement.

The layout of this paper is as follows: in §2, we briefly review the **IPF** data structure and its usage [3], and describe a coherent framework for multi-layer region selection in **IPFs**; in §3, we describe the spine identification algorithm itself; in §4 we present our results; and in §5 we outline our future plans and conclude.



(a) A simple striped image

(b) One possible partition forest for it: solid lines are part of a tree; dashed lines are edges in the **RAG** for a particular layer of the tree

Figure 1. Partition Forest Example

2. Image Partition Forests (IPFs)

2.1. Structure

The structure of an image partition forest (see Figure 1) can be defined as follows:

Definition 1 *Given a region $\mathcal{R}$ of an image, a partition* $\mathbf{P}(\mathcal{R})$ *of $\mathcal{R}$ is a set of sub-regions $\{R_1, \ldots, R_r\}$ such that $\bigcup_i R_i = \mathcal{R}$ and $\forall i, j \cdot i \neq j \Rightarrow R_i \cap R_j = \emptyset$.*

Definition 2 *Given a region set $R = \{R_1, \ldots, R_r\}$, a region adjacency graph* $\mathbf{RAG}(R)$ *of R is a graph with nodes R_i and edges defined by $w : R \times R \rightarrow \mathbb{R}^+$. There is an edge between any given pair of regions R_i and R_j iff $w(R_i, R_j) \neq \infty$, in which case $w(R_i, R_j)$ gives the weight on the edge.*

Definition 3 *A partition tree* $\mathbf{PT}(\mathcal{R})$ *of $\mathcal{R}$ is a tree where each layer corresponds to a partition of $\mathcal{R}$. Each node in the tree represents a sub-region of $\mathcal{R}$, with the root node representing $\mathcal{R}$ itself. The sub-region represented by any non-leaf node (i.e. any node not in the lowest tree layer) is the union of the sub-regions represented by its children. In addition to the tree itself, we maintain a* **RAG** *for each layer of the tree.*

Definition 4 *Given a partition* $\mathbf{P}(I) = \{R_1, \ldots, R_n\}$ *of an image, a partition forest* $\mathbf{PF}(I)$ *of the image is a set of partition trees, one for each R_i. Note that there is no unique partition* $\mathbf{P}(I)$ *of an image. Likewise, there is no unique partition tree for any region in a given* $\mathbf{P}(I)$*. For that reason, there are any number of different partition forests for the same image.*

Since partition forests for an image are not unique, it is important to choose a construction algorithm that produces a partition forest which is useful for a given problem domain. In the case of our segmentation work, we found that a combination of the watershed and waterfall algorithms [2, 6] could be used to produce a suitable initial partition forest (in practice, a single partition tree, which is then suitable for further refinement by the user).

2.2. Usage

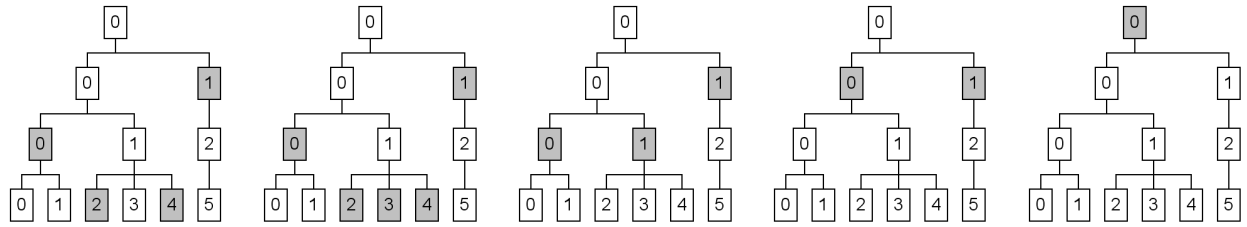
As we described in [3], the nodes in a partition forest can be annotated efficiently with useful properties of the sub-regions of the image they represent. This enables quick (linear in the number of nodes) searches for regions satisfying certain properties. Since we have adjacency information for each layer in the forest in the form of a **RAG**, these searches can be quite intricate; for example, we can search for regions which are small, have high max and mean grey values, are moderately elongated and are surrounded by darker regions (these make good candidates for ribs).

Because the initial construction of a partition forest is based purely on the grey levels in an image, it is useful to introduce anatomical knowledge to refine the forest at a later stage. As reported in [3], there are various useful operations that can be performed on the forest, including splitting and merging regions, and moving a subtree in the partition to be a child of a different parent node in the layer above its root. It is also important that certain regions in the tree can be identified as being particular features of interest, e.g. a kidney. These operations are non-trivial in that they require careful changes to the partition forest, but can be done in an efficient way (including updating node annotations) as we described [3].

2.3. Multi-Layer Selection

In order to facilitate easy editing of the partition forest, it is important to support the selection of forest nodes. Allowing the user or an algorithm to refer to a single node in the forest is evidently a simple process: all that is necessary is to define a suitable system of node identifiers (e.g. in our work, we use identifiers of the form $(layer, index)$ to refer to nodes in the forest: we number nodes in layers rather than by tree — see Figure 1). Further, each parent node in the tree represents the union of the sub-regions represented by its children; for example, in Figure 1, selecting node $(1, 2)$ is equivalent to selecting nodes $(0, 5)$ and $(0, 6)$.

This alternative viewpoint is the motivation behind the useful idea of multi-layer node selection. Letting $\mathbf{PS}(I)$ be the pixel set of image I, and the current selection be a set



(a) Initial selection (grey) (b) Adding $(0, 3)$ (c) Consolidating $(1, 1)$ (d) Consolidating $(2, 0)$ (e) Consolidating $(3, 0)$

Figure 2. Adding a node to a multi-layer selection

of pixels $S \subseteq \mathbf{PS}(I)$, we define a minimal node representation of S, which we will denote as $\mathbf{MNR}(S)$, as the smallest set of nodes that represents S. This implies that we select a parent node rather than its children individually; for example in Figure 1, consider the selection S of all pixels in $(0, 2)$, $(0, 3)$ and $(0, 4)$, then $\mathbf{MNR}(S) = \{(0, 2), (1, 1)\}$, and if we removed the pixels in $(0, 3)$ from the selection $\mathbf{MNR}(S)$ would become $\{(0, 2), (0, 4)\}$. Such a minimal representation will naturally give rise to nodes at different levels, and hence the term multi-layer selection.

Having explained this concept we now turn to how to implement it using the minimal node representation described. In particular, we focus on the two key operations involved, adding and removing nodes. From an end-user perspective, it is helpful if the operations can be made undoable: this can easily be accomplished by recording which node identifiers are added and removed from **MNR**(S) by each operation.

Adding a Node There are four cases to deal with when adding a node to the selection:

1. The node is already in **MNR**(S) $\Rightarrow$ do nothing.
2. An ancestor of the node is already in **MNR**(S) $\Rightarrow$ do nothing.
3. One or more descendants of the node are already in **MNR**(S): we remove them and add this node instead, since it contains all its descendants.
4. Otherwise: simply add the node to **MNR**(S) directly.

If we added a node identifier (either of the latter two cases), we need to *consolidate* the selection as a final step: this involves replacing any node whose children are all selected with the node itself in **MNR**(S). We only need to consider nodes which might have been affected by the new node addition when performing this process: thus, it suffices to recursively consolidate nodes from the parent of the added node upwards. If at any stage one of the children of the current node is not part of the selection, the consolidation process terminates; otherwise, we recurse on the parent of the node we just consolidated. See Figure 2 for an example.

Removing a Node There are four cases to deal with when removing a node from the selection as well:

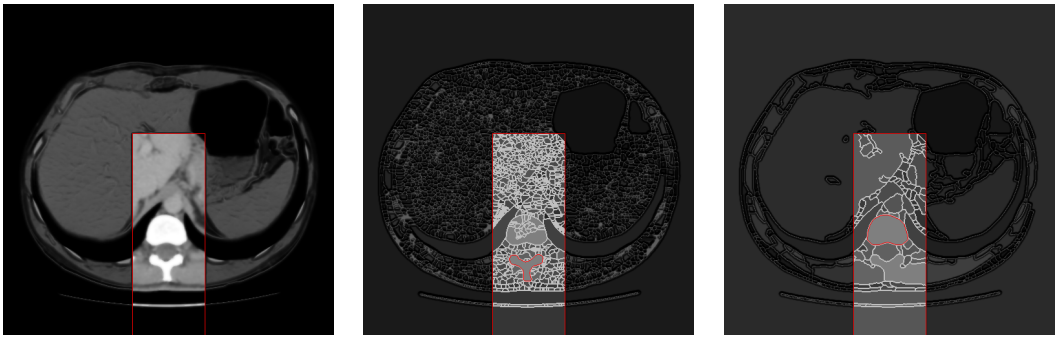
1. The node is in **MNR**(S) $\Rightarrow$ remove it.
2. An ancestor of the node is in **MNR**(S); this case will be covered below.
3. Some descendants of the node are in **MNR**(S) $\Rightarrow$ remove them.
4. Otherwise no part of S lies within the region represented by the node, so do nothing.

The interesting case involves removing a node whose ancestor is in **MNR**(S). To do this, we find the *trail* of nodes in the tree leading from the ancestor to the node itself. We then recursively split the nodes along this trail until we get back to the original node: the intermediate **MNR**(S) then contains the original node and we can simply remove it. For example, consider Figure 2 in reverse (i.e. consider removing $(0, 3)$ from the selection in (e)). Here, the trail of nodes from the ancestor (namely $(3, 0)$) is $(3, 0)$, $(2, 0)$, $(1, 1)$. Each of these should be split in turn until the representation contains the grey nodes in (b): at this point, removing $(0, 3)$ is trivial.

3. Spine Identification

Our spine identification algorithm (see Figure 4) works by doing region growing on the region adjacency graphs stored with the image's partition forest (note that this approach differs from various region growing segmentation algorithms which work only on the pixels of an image – e.g. [4, 7]). The outline of the algorithm is as follows (details are given later):

1. **Seed Finding**. Traverse the forest to find regions which satisfy a user-specified *seed criterion*.
2. **Region Flooding**. Determine a *preliminary feature* (represented as a multi-layer selection) by region flooding from the various seeds.
   (a) Construct a map to indicate whether or not each seed has yet been visited during the flooding process (each seed is initially marked as unvisited).



(a) The region to search (b) A spine seed in partition 0 (c) A spine seed in partition 1

Figure 3. Finding suitable spine seeds: it suffices to look for moderately-sized, bright regions with centroids within the box shown

(b) For each seed:

i. If the seed has not already been visited, identify a *potential partial feature* for it by performing a flood from it in the **RAG** for its layer. The flooding process is controlled by a user-specified *flooding criterion*.

ii. If the potential partial feature satisfies a user-specified *validation criterion*, add the regions it contains to the preliminary feature.

iii. Mark any seeds which were reached from the current seed as having been visited.

3. **Post-Processing**. Remove any regions which were undesirably added by the flooding process (the regions to be removed are selected using a user-specified *removal criterion*).

(a) For each region in the **MNR** of the preliminary feature:

i. If the region satisfies the removal criterion, remove it using the appropriate algorithm for multi-layer selections described earlier.

ii. Otherwise, recurse on its children (if any).

It is worth noting that this is a general scheme that can be used for features other than the spine, provided that suitable seed, flooding, validation and removal criteria can be specified in each case. The role of multi-layer selection in the process is two-fold: it is used to union the partial features identified in the second phase into the preliminary feature, and it is then used to facilitate modifications to the feature by the post-processing phase.

3.1. Seed Finding

The criterion we use to find suitable spine seeds (in a 512x512 image) is that regions should be moderately-sized (area ≥ 500 pixels), fairly bright (grey value mean ≥ 190, where 255 is white) and roughly centred in the bottom-middle of the image ($200 \leq x \leq 312$ and $y \geq 200$) – see Figure 3. This is sufficient in the case of the spine, since the only other features of sufficient brightness in an image (the ribs) are either smaller than the specified threshold, or not centred within the specified box.

3.2. Region Flooding

As mentioned earlier, a region flooding process is used to determine a potential partial feature for each as yet unvisited seed. In each case, flooding starts from the seed and works recursively: to flood from a region r, we consider as yet unseen regions adjacent to r (in the **RAG** of the seed's layer) and test them using the flooding criterion. Iff a region satisfies the criterion, we add it to the potential partial feature for the seed and recurse on it. (In either case, we mark the tested region as seen to ensure that it is never tested again.)

The flooding criterion itself can be arbitrarily complicated (e.g. it can depend on the properties of the current and adjacent regions, and on those of the current partial feature as a whole, if necessary), but in our case we found it sufficient to use a criterion which ensured that the adjacent region's grey value mean was > 170 and that its centroid satisfied $200 \leq x \leq 312$ and $y \geq 256$.

As remarked above, each partial feature is validated to ensure that it should form part of the preliminary spine, and added to it if it passes. In our implementation, the validation criterion simply ensures that the centroid of the partial feature is appropriately in the bottom-centre of the image ($200 \leq x \leq 312$ and $y \geq 256$).

3.3. Post-Processing

The preliminary spine generated by the region flooding process is generally fairly good, but it's possible that in some cases we added regions we didn't intend. In particular, this can happen because either: (a) we added a large region whose grey value mean was high, but there were smaller regions contained within it with a low grey value mean; or (b) we had to set the grey value mean tolerance in our flooding criterion sufficiently low to capture greyer bits of the spine,

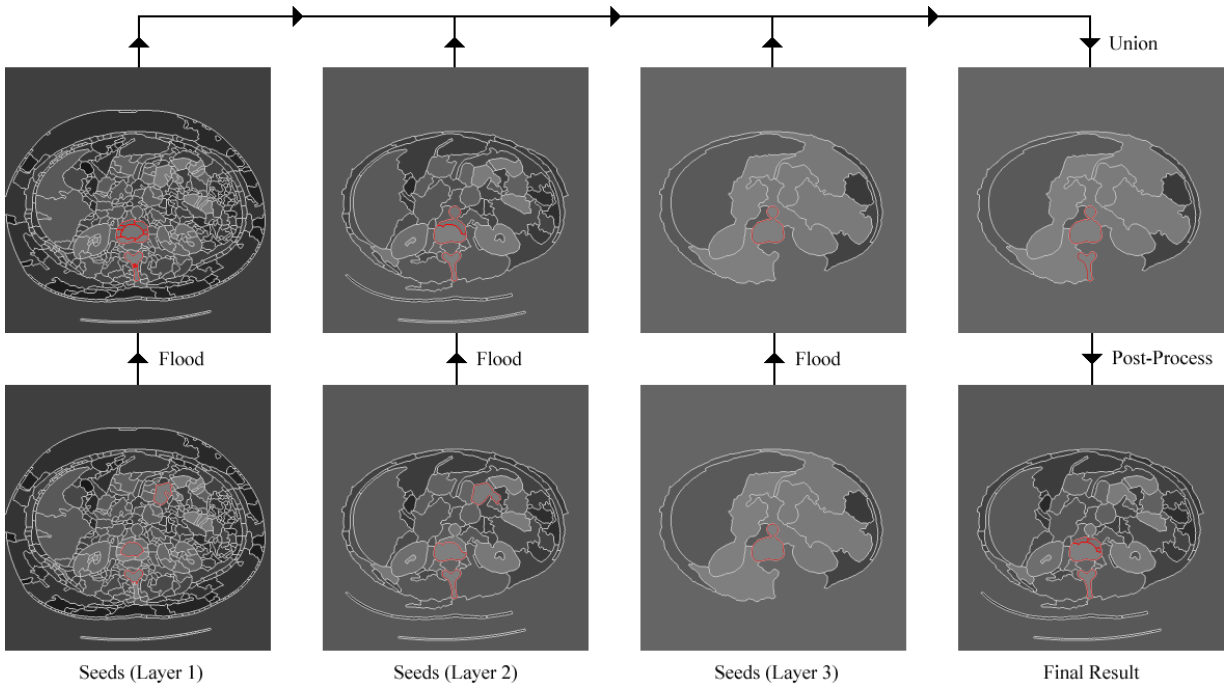


Figure 4. The spine identification process (see text)

but that caused us to inadvertently capture bright grey things like the aorta (which can be quite close to the spine in some images) as well.

We therefore post-process the results of the flooding to attempt to remove these undesirable features. The process proceeds down each selected subtree, looking for regions to be removed. In the case of our spine identifier, our removal criterion had two different cases, to deal with each of the two issues identified above. In the case of (a), we found that removing dark regions (grey value mean < 150) of a reasonable size (area ≥ 400) worked well. For (b), we simply removed things which looked like the aorta ($400 \leq$ area ≤ 700 and $160 \leq$ grey value mean ≤ 180 and elongatedness < 1.5). (A definition of elongatedness appears in [3].)

4. Results

We tested our spine identification algorithm on images from a number of series (see Figure 5 for some examples) and it proved quite robust. It can still fail sometimes in cases where the spine appears significantly darker than usual (e.g. MC-2-135) or where there are small pieces of spine which are disconnected from the primary feature (e.g. MC-2-136 and MC-2-137), but in the overwhelming majority (87.3%) of the cases we tested it performed well, as seen in Table 1 (in which A denotes a perfect result for a slice, B an almost perfect one, C an adequate one, and F a failure). It is worth noting that since the result is presented visually to the user (e.g. a radiologist) as a multi-layer selection, it is very easy for them to verify the output and make alterations where desired.

Series	A	B	C	F	A/B	Total	%A/B
BAH-2	11	4	1	1	15	17	88.2
BT-2	25	12	3	1	37	41	90.2
EJ-2	18	2	1	0	20	21	95.2
MC-2	6	2	1	2	8	11	72.7
MJ-2	10	5	0	6	15	21	71.4
RC-2	7	5	0	0	12	12	100.0
SD-2	7	3	1	0	10	11	90.9
All	84	33	7	10	117	134	87.3

Table 1. Spine identification results (ordered alphabetically by series). See Figure 5 for sample images.

5. Conclusions and Future Work

In this paper, we presented an automatic approach to spine identification in abdominal CT images based on region flooding over the region adjacency graphs associated with a partition forest for an image. As illustrated, this yielded good results on images from a number of different series. We also presented a general approach to multi-layer selection in partition hierarchies: this is useful in many other contexts besides automatic segmentation (in particular, we have found it useful for managing meta-data stored hierarchically in a database).

Future work will aim to build a system to automatically define a robust frame of reference (for localization purposes) on each CT slice, based on the spine identification algorithm presented here, and our previous work on rib idenitification [3]. We also feel that the flooding algorithm

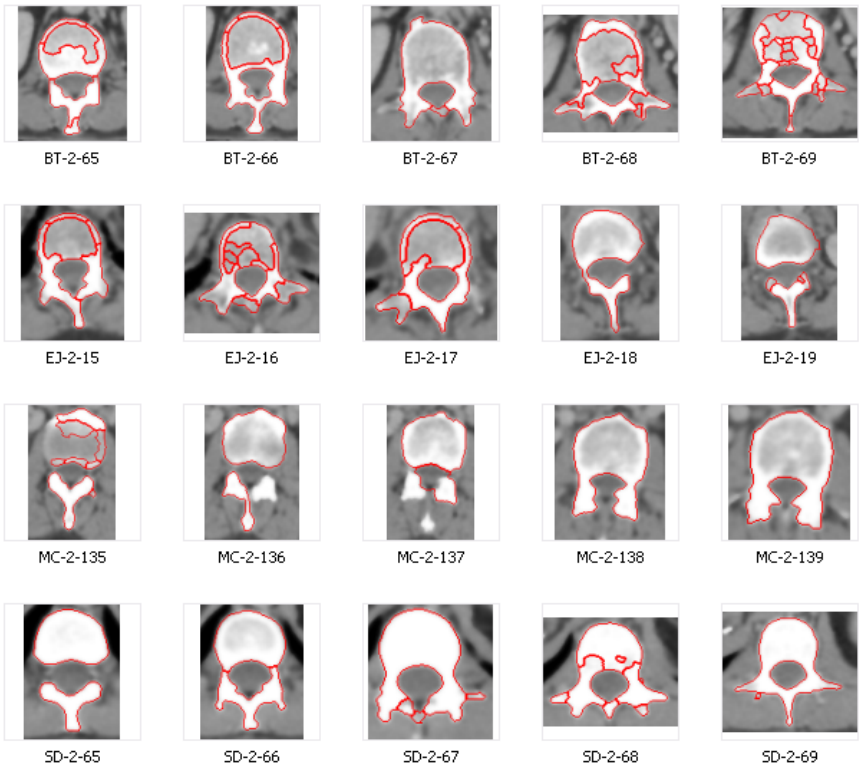


Figure 5. Results of the spine identification algorithm on images from four different series (BT-2, EJ-2, MC-2 and SD-2): each sub-region is surrounded by its own red border, thus double-thickness borders indicate an internal boundary rather than the boundary of the feature. See Table 1 for comprehensive statistical results by series.

serves as a useful template on which to build more intricate automatic identifiers for other features, and plan to study this further.

6. Acknowledgements

We would like to thank medics and technicians from the Churchill Hospital, Oxford, for their invaluable help and support: Zoë Traill Consultant Radiologist, David Cranston Consultant Urological Surgeon, Mark Sullivan Consultant Urological Surgeon, Andrew Protheroe Consultant Medical Oncologist, Anthony McIntyre Superintendent Radiographer, Nilay Patel Specialist Registrar, Rob Ritchie Academic Clinical Fellow in Urology. We are also grateful to Clarice Poon who carried out some of the statistical analysis.

References

[1] N. Archip, P.-J. Erard, M. Egmont-Petersen, J.-M. Haefliger, and J.-F. Germond, "A Knowledge-Based Approach to Automatic Detection of the Spinal Cord in CT Images", *IEEE Transactions on Medical Imaging*, 21(12), December 2002.

[2] S. Beucher, "Watershed, hierarchical segmentation and waterfall algorithm", In *Proceedings of ISMM'94*, pp. 69–76, 1994.

[3] S. Golodetz, I. Voiculescu, and S. Cameron, "Region Analysis of Abdominal CT Scans using Image Partition Forests", In *Proceedings of CSTST'08*, pp. 432–7, October 2008.

[4] D.-T. Lin, C.-C. Lei, and S.-W. Hung, "Computer-Aided Kidney Segmentation on Abdominal CT Images", *IEEE Transactions on Information Technology in Biomedicine*, 10(1), January 2006, pp. 59–65.

[5] R. Lu and P. Marziliano, "Liver Tumor Volume Estimation by Semi-Automatic Segmentation Method", In *Proceedings of the 2005 IEEE Engineering in Medicine and Biology 27th Annual Conference*, 2005.

[6] B. Marcotegui and S. Beucher, Fast Implementation of Waterfall Based on Graphs, In *Mathematical Morphology: 40 Years On*, Springer Netherlands, 2005.

[7] R. Pohle and K. Toennies, "Segmentation of Medical Images Using Adaptive Region Growing", In *Proceedings of SPIE (Medical Imaging)*, volume 4322, pp. 1337–1346, 2001.

[8] K.-S. Seo, L. C. Ludeman, S.-J. Park, and J.-A. Park, "Efficient Liver Segmentation Based on the Spine", *Springer Lecture Notes in Computer Science*, 2004, pp. 400–9.

[9] Z. Wu and J. M. S. Jr., "Multiple material marching cubes algorithm", *International Journal for Numerical Methods in Engineering*, 58(2), July 2003, pp. 189–207.

[10] J. Yao, S. D. O'Connor, and R. M. Summers, "Automated Spinal Column Extraction and Partitioning", In *3rd IEEE International Symposium on Biomedical Imaging: Nano to Macro*, pp. 390–393, 2006.