3D Scene Reconstruction Using Partial RGB+Depth Maps

Matej Kopernický[1]*

Supervised by:
Ing. Radoslav Gargalík[2]†
Mgr. Tomáš Kovačovský[1]‡

[1]Faculty of Mathematics, Physics and Informatics
Comenius University
Bratislava / Slovakia

[2]Faculty of Science
Pavol Jozef Šafárik University
Košice / Slovakia

Abstract

We present a complex approach to 3D scene reconstruction using both RGB images and depth maps provided by a Microsoft Kinect sensor. The sequence of frames recorded by the freely moving camera fully controlled by human user is processed in order to obtain point cloud representation of the recorded scene. An ICP method is used to align point clouds of subsequent frames, using matched SURF descriptors extracted from RGB images as the initial alignment estimate. Nearest neighbour search for an ICP is accelerated by a kd-tree. The alignment of subsequent frames allows to fully reconstruct camera movement in 3D space and thus to create a complex, colored point cloud model of the recorded scene or object.

Keywords: 3D reconstruction, Microsoft Kinect, point cloud, ICP, SURF, k-d tree, quaternions

1 Introduction

In recent years, the advance in portable computers and micro-electronics brings demand for 3D content, real-time augmented reality applications and natural human-computer interfaces like gesture control. It is therefore necessary to efficiently capture the shape, color or other properties of the objects or the whole scene, what often includes the need of 3D reconstruction.

Currently, there are multiple 3D reconstruction methods available, varying from the active approaches including mechanical contact with the object or laser scanning, to the passive methods able to create the scene model from the video or multiple photographs. The 3D reconstruction is usually intended for professional use due to high price of required hardware and its difficult manipulation. However, in recent years, multiple cheap and widely accessible devices have been introduced, especially motion input sensing devices like *Microsoft Kinect* or *Asus Xtion*. Thanks to their ability to simultaneously capture RGB images and depth maps, they have become a popular tool for the 3D reconstruction.

The most notable work using such devices for the 3D reconstruction is the *Kinect Fusion* [10][13], where authors have presented a complex approach to the real-time 3D scene reconstruction using depth maps provided by Kinect. The detailed reconstructed model provides an interaction with the user, allowing him to interact with individual objects of the scene.

In this work, we will present a complex method for 3D scene reconstruction which uses both the RGB images and depth maps, captured by freely moving camera controlled by user, in contradiction to the usual approach taking only depth maps into consideration. This can bring an advantage in cases, when the standard approach would not work efficiently or would not work at all.

2 Rigid body registration

The data obtained by an input device can be viewed as a sequence of frames, where each frame consists of an RGB image and the corresponding point cloud obtained from the depth map. Our goal is to create the 3D model representing the scene captured by the frame sequence. In order to incorporate multiple point clouds into into a single one, relative transformations between subsequent frames has to be known. This is called the rigid body registration problem. The basic outline of this approach is described in the Algorithm 1.

To estimate the transformation between two frames, the corresponding point pairs have to be found. They are the points of two point clouds that represent the same spot of the scene. The searching methods of the corresponding point pairs are described in Sections 2.1 and 2.2. After obtaining corresponding pairs, transformation can be estimated, as presented in Section 2.3. The idea of the transformation estimate is depicted on Figure 1.

*matej.kopernicky@gmail.com
†radoslavgargalik@gmail.com
‡tomas.kovacovsky@gmail.com

Algorithm 1: 3D scene reconstruction algorithm outline

Input: Sequence of recorded frames $\mathcal{F}_1, \ldots, \mathcal{F}_n$
Output: Resulting point cloud $\mathcal{R}$ representing 3D model of the recorded scene

```
// Total transformation representing
   camera position change between
   the first frame and the current
   frame in each iteration
```
$T_{total} \leftarrow Identity$;
```
// Initially, the resulting model
   contains only data from the first
   frame.
```
$\mathcal{R} \leftarrow \mathcal{F}_1$;
for $i \leftarrow 2$ *to* n **do**
  $T \leftarrow$ transformation estimate that aligns the point cloud of $\mathcal{F}_i$ with the point cloud of $\mathcal{F}_{i-1}$;
  ```
  // composition of two spatial
     transformations
  ```
  $T_{total} \leftarrow T + T_{total}$;
  $\mathcal{F}_i \leftarrow T_{total}(\mathcal{F}_i)$;
  $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{F}_i$;
end
return $\mathcal{R}$;

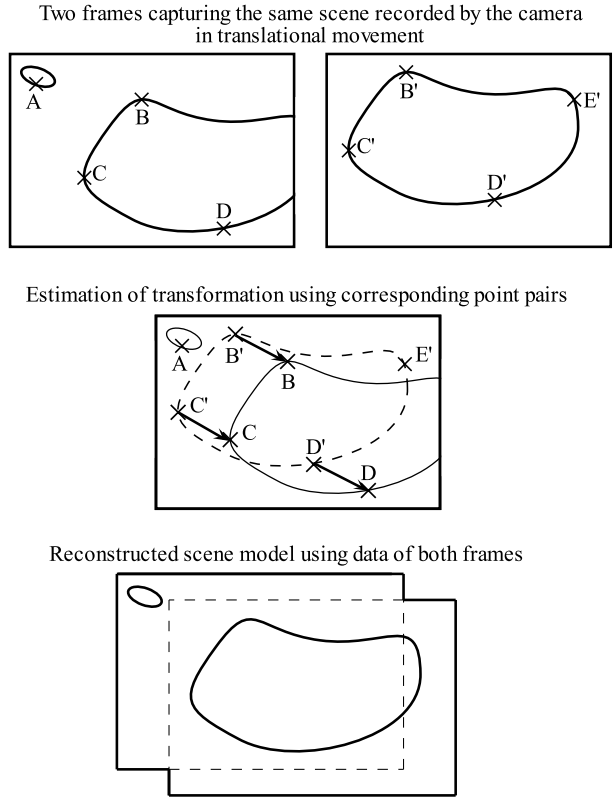

Figure 1: A simplified model reconstruction example in 2D space, using two frames.

2.1 ICP

The ICP (Iterative Closest Point) method [3] is based on a simple idea of iterative distance minimization between two frames. For each point from the first point cloud, the nearest neighbour in the second cloud is found, establishing a correspondence between them. These point pairs will in most cases not represent the same physical point of the scene. However, considering many iterations and thousands of corresponding pairs, we are able to align two point clouds by obtained transformation estimate. The implementation of the ICP algorithm is included in the final Algorithm 2 describing the whole process of the rigid body registration.

2.2 Initial estimation

Since the ICP minimizes distance between two point clouds, this approach is not sufficient in all cases. Consider the camera moving along the wall (Figure 2), which forms, regarding the spatial distribution of points in the point cloud, a plane in the 3D space. Distance minimization between two point clouds obtained from such a frame sequence would neglect the translational movement of the camera, which would lead to severe inaccuracies in the resulting model (Figure 3). For this reason, an initial transformation estimate is often made before applying the ICP itself. We will use RGB images to make the first step when aligning two point clouds.

An appropriate method allowing to establish correspondences between points of two frames is the SURF (Speeded Up Robust Features) [1]. This feature extractor is able to detect specific, characteristic parts of the RGB image – edges, corners, blobs or ridges. These are usually described by a 64-dimensional descriptor vector. Thanks to the method's invariance to rotation, scaling, viewpoint change, blurring and other image transformations [8], it is possible to detect the same features on two subsequent RGB images of the input sequence. The obtained descriptors from two images are paired by searching for the closest neighbours again, this time in the 64-dimensional descriptor space using Euclidean distance. After obtaining corresponding pairs on two RGB images, they are associated to the points of the point clouds. This can be used to effectively estimate the transformation, giving the initial step before applying the ICP. This not only helps to avoid situations when two point clouds are incorrectly aligned (Figure 3), but also can reduce the required number of ICP iterations, speeding up the reconstruction process.

2.3 Absolute orientation problem

Having established associated point pairs, a transformation approximating spatial transformation between two point clouds can be found. Multiple methods

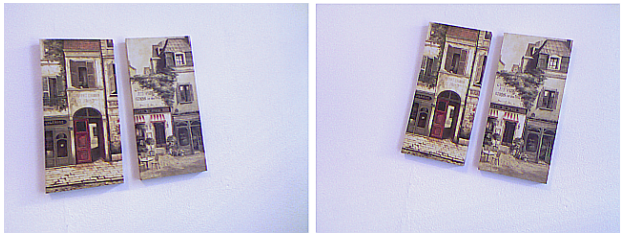

Figure 2: Two frames of the sequence recorded by the camera moving along the wall.

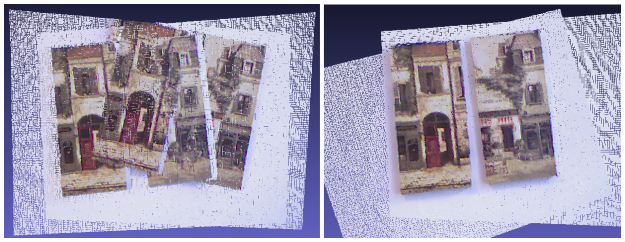

Figure 3: Incorrect alignment using only ICP (left) and correct alignment with the initial estimation using SURF method (right).

have been described and compared considering estimate precision and computational efficiency [5][11]. In this work, the Horn's method [9] has been chosen due to satisfying results comparable with other known methods, good description in numerous other works in the field of 3D reconstruction and its simple implementation in our resulting application. Chosen method exploits properties of quaternions and minimizes mean square error of transformation estimation. We will briefly explain key steps of this method and present the resulting algorithm, taking into account specifics of our approach to 3D reconstruction.

Consider two sets of associated points in 3D space $\mathcal{A} = \{a_1, \ldots, a_n\}$ and $\mathcal{B} = \{b_1, \ldots, b_n\}$, such that $a_i \in \mathcal{A}$ forms a corresponding pair with $b_i \in \mathcal{B}$ $\forall i = 1, \ldots, n$. We search for the best transformation estimate T such that $\mathcal{B} = T(\mathcal{A}) = R(\mathcal{A}) + \vec{t}$. Error estimation for a point pair can be expressed as $e_i = b_i - R(a_i) - \vec{t}$ $\forall i = 1, \ldots, n$. Our goal is to minimize mean square error, which means we search for a transformation T minimizing $\sum_{i=1}^{n} \|e_i\|^2$.

The key idea of Horn's method is based on rotation represented by quaternion. Quaternions are an "extension" of complex numbers by another two imaginary parts. By reffering two sets of associated points to their respective centroids and by using useful quaternion properties, Horn has shown that the transformation R is given by the eigenvector corresponding to the maximal eigenvalue of the matrix

$$\begin{bmatrix} S_{xx}+S_{yy}+S_{zz} & S_{yz}-S_{zy} & S_{zx}-S_{xz} & S_{xy}-S_{yx} \\ S_{yz}-S_{zy} & S_{xx}-S_{yy}-S_{zz} & S_{xy}+S_{yx} & S_{zx}+S_{xz} \\ S_{zx}-S_{xz} & S_{xy}+S_{yx} & -S_{xx}+S_{yy}-S_{zz} & S_{yz}-S_{zy} \\ S_{xy}-S_{yx} & S_{zx}+S_{xz} & S_{yz}+S_{zy} & -S_{xx}-S_{yy}+S_{zz} \end{bmatrix}$$

Here, $S_{\alpha\beta} = \sum_{i=1}^{n} a'_{i_\alpha} b'_{i_\beta}$, where a'_{i_α} and b'_{i_β} represent x, y or z-coordinates of the input points from $\mathcal{A}$ and $\mathcal{B}$ referred to their centroids. After that, the translational component of the transformation can be easily found by substituing the found rotation R in the original equation $\mathcal{B} = R(\mathcal{A}) + \vec{t}$.

2.3.1 Transformation composition

Storing all the transformations found in the individual steps of the algorithm and then successively transforming all the points of the point cloud would be inefficient. Since transformation estimate consists of rotation and translation, it does not represent linear transformation and thereby cannot be composed by the simple matrix multiplication. Using homogenous coordinates, the total transformation T composed from $T_1, \ldots, T_n$ can be represented by 4×4 matrix

$$M_T = \begin{bmatrix} a & b & c & \vec{t}_x \\ d & e & f & \vec{t}_y \\ g & h & i & \vec{t}_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

whose elements $\vec{t}_x, \vec{t}_y, \vec{t}_z$ and $a, \ldots, i$ are to be found. Consider points $v_0 = (0,0,0,0)$, $v_x = (1,0,0,0)$, $v_y = (0,1,0,0)$, $v_z = (0,0,1,0)$, which are successively transformed by each of T_i for $i = 1, \ldots, n$ during individual iteration of the algorithm and $T(v_0), T(v_x), T(v_y), T(v_z)$ are obtained in the end. We know that $M_T v_0 = T(v_0)$, $M_T v_x = T(v_x)$, $M_T v_y = T(v_y)$, $M_T v_z = T(v_z)$ and

$$\begin{aligned} T(v_0) &= (\vec{t}_x, \vec{t}_y, \vec{t}_z, 1) \\ T(v_x) &= (a + \vec{t}_x, d + \vec{t}_y, g + \vec{t}_z, 1) \\ T(v_y) &= (b + \vec{t}_x, e + \vec{t}_y, h + \vec{t}_z, 1) \\ T(v_z) &= (c + \vec{t}_x, f + \vec{t}_y, i + \vec{t}_z, 1) \end{aligned}$$

Elements of matrix M_T can be thus easily found using resulting coordinates of only four additional points, which are successively transformed in each iteration of our algorithm.

The resulting, complete iterative algorithm finding the transformation estimate between two succesive frames from the input sequence is described in Algorithm 2.

3 Implementation

All data have been provided by Microsoft Kinect camera, which represents a simple and very cheap (currently with a €100 price) source of data for the 3D reconstruction and many other applications in the augmented reality and especially gaming industry. Kinect provides the standard RGB camera with 640×480 resolution and the infared

Algorithm 2: Estimation of the transformation aligning two frames

Input: D_1, D_2, RGB_1, RGB_2 : Depth maps and corresponding RGB images of two frames
ICP_{max} : maximum number of ICP steps
err_{limit} : acceptable mean squared error of transformation estimate

Output: T : transformation aligning two frames

```
// SURF descriptors described in Section 2.2
desc1 ← find_desccriptors(RGB1);
desc2 ← find_desccriptors(RGB2);
pts1, pts2 ← empty point lists ; // list of associated point pairs
foreach d in desc2 do
    closest ← closest_desc(d, desc1);
    // reversed descriptor correspondence check
    if closest_desc(closest, desc2) = d then
        pts2 ← pts2 ∪ get3Dpoint (d, D2);
        pts1 ← pts1 ∪ get3Dpoint (closest, D1);
    end
end
// initial estimation using Horn's quaternion method
T ← find_transform(pts2, pts1);
pts ← T(get3Dpoints(H2));
v0, vx, vy, vz ←
T(0,0,0), T(1,0,0), T(0,1,0), T(0,0,1);
kd ←build_kd(get3Dpoints(H1));
for i ← 1 to ICPmax do
    // find closest point pairs
    pts1, pts2 ← empty point lists;
    foreach p in pts do
        neighbour ← closest_neig(p, kd);
        pts2 ← pts2 ∪ p;
        pts1 ← pts1 ∪ neighbour;
    end
    // Horn's quaternion method
    T ← find_transform(pts2, pts1);
    pts ← T(pts);
    v0, vx, vy, vz ← T(v0), T(vx), T(vy), T(vz);
    if estimate_err(pts1, T(pts2)) < errlimit then
        break;
    end
end
// find total transformation
T.translation ← (v0[1], v0[2], v0[3]);
T.rotation ←
[vx[1]−v0[1]  vy[1]−v0[1]  vz[1]−v0[1]
 vx[2]−v0[2]  vy[2]−v0[2]  vz[2]−v0[2]
 vx[3]−v0[3]  vy[3]−v0[3]  vz[3]−v0[3]];
return T;
```

sensors capable of producing 320×240 depth map of the scene. Since in movement both RGB pictures and depth maps tend to contain more noise and inaccuracies, the individual frames of the input sequence were rather captured by a stable camera (like photographs).

A presented method for the 3D reconstruction have been implemented in *C#* running on *.NET4*, using Kinect SDK (Software Development Kit). Multiple applications with the user-friendly interface have been created, providing scene recording and storing for later processing, removing superfluous frames, and the reconstruction itself with the possibility of setting reconstruction parameters. To achieve good performance and ensure needs of our application, the most of the code have been written form the very beginning, including the kd-tree (see below) and Horn's quaternion method. For SURF features detection *OpenSURF* library have been used and for the matrix manipulation and eigenvalue/eigenvector search we have chosen *Math.NET* library.

3.1 Nearest neighbour search

With Microsoft Kinect, it is possible to obtain $320 \times 240 = 76800$ points for each frame, while many other devices are able to produce more detailed depth maps. It would be then very inefficient to search for the nearest neighbours for ICP using only brute force approach. It is then necessary to speed up the search process, which can be achieved by a suitable space partitioning data structure – a kd-tree [2][7].

Kd-tree (k-dimensional tree) is a binary tree, where each node represents one point of the input point cloud. This data structure splits space into two half-spaces by a hyperplane perpendicular to one of the dimensions' axis in each non-leaf node. Points located in those two half-spaces form two subtrees of the node. The axis regularly alternates for each level of the tree using the same pattern until reaching the last level with leaf nodes. It is proven in [12] that the nearest neighbour query works in $O(kn^{1-1/k})$, where k is the number of dimensions and n the number of points stored in the tree. In our case of $k = 3$, the complexity is then $O(3n^{2/3})$. However, in most of the queries with real data, the number of nodes to check is significantly lower than the upper bound.

In order to speed up kd-tree traversal, the tree should be balanced, having at most $\lceil log(n) \rceil$ levels where n is the total number of points stored in the tree. This can be achieved by selecting points in a way that subtrees of the point's node will have approximately the same size. It is therefore required to find a median of the points with respect to the dimension defined by the splitting hyperplane. First, three lists of points are created, each sorted by one of the points' dimension, which can be achieved in $O(n * log(n))$ time. Then in each node, a

median can be easily found by simply picking the point in the middle of the right list in $O(1)$ time. New point lists are then created for both subtrees, preserving the order from current point list (three lists for both subtrees, each in $O(\frac{n}{2})$) and subtrees are then recursively constructed. Analyzing time complexity $T(n)$ of the (sub)tree construction in each of the node, having n points, we obtain $T(n) = O(1) + 6O(\frac{n}{2}) + 2T(\frac{n}{2}) = O(n) + 2T(\frac{n}{2})$, resulting in a total $O(n * log(n))$ complexity of kd-tree construction, which can be proven by master theorem [4].

Constructed kd-tree can be then used for an efficient closest neighbour search of the point P:

1. First, an initial estimate is made by recursively traversing the tree from its root in logarithmic time, always choosing the subtree which ensloses half-space where P is located until reaching a leaf node.

2. Second tree traversal uses the estimate obtained in previous phase. All subtrees that can possibly contain some point with a distance to P lower than the first estimate are checked, while the others are skipped.

In addition to standard approach, each node stores boundaries of the space part where its subtree points can be located. When building the tree, the node stores not only point itself, but also the information about how the space was partitioned in its parent nodes. This allows to omit subtrees that would be checked using the standard approach.

4 Results

Since Kinect sensors allow capturing depth map data only in $0.8 - 4m$ distance range, our method has been tested mainly in the interior environment, allowing us to reconstruct room interiors, furniture or small and medium sized objects. The additional challenge to deal with is given by the low resolution of depth maps and constant amount of noise, which does not allow to describe the scene with all of its details. However, our proposed method itself can be applied widely for many different data sources of varying character and quality.

Our test have shown the capability of the presented method to create scene models preserving its geometry and correctly mapping the "texture" (assigning color to each point of the resulting point cloud). Without the initial step, (see Section 2.2), the ICP required from 20 to 50 iterations (depending on the scene) to align two point clouds. With the initital step, this number decreased to 6-20 and in addition, it provided better alignment for some cases. The alignment of two frames has taken from one to two seconds on the mediocre hardware (Intel Core2Duo E7200 @ 3.5GHz with 4GB RAM). Time requirements of the individual parts of the reconstruction algoritm were tested on the kitchen input set consisting of 23 frames and are shown in Figure 5.

5 Conclusions

We have presented and practically tested a method for the 3D reconstruction leading to the colored point cloud scene model. The input consists of the sequence of corresponding RGB pictures and depth maps captured by the user-controlled, freely moving camera. Since method uses both RGB pictures and depth maps in the process of reconstruction, it can be applicable in conditions when many other methods using only geometry of the scene would fail.

Using only cheap and common hardware, we have provided the simple and accessible 3D reconstruction tool even for non-professional users in the home environment. Thanks to its ability to create models of room interiors or medium-sized objects, it can become a good starting point to the 3D modeling, providing basic geometry of a scene or object that can be further manually corrected and enhanced.

The created application can be further accelerated by lower-level programming using C++ language or massive parallelization provided by GPU computing. This could lead to real-time 3D reconstruction with an on-fly model creation during the recording itself. Another improvement could be based on more advanced variations of the ICP method or on a different initial transformation estimate.

Figure 4: Reconstructed 3D model of the balcony scene. The irregularities of the wall color are caused by varying light conditions when capturing the individual frames.

	Total time [ms]	Time per frame [ms]
SURF pairing	6906	310
Initial transformation estimation	60	2
Kd-tree construction	7137	310
Nearest neighbour search (all 8 ICP iterations)	23432	1065
Transformation estimation (all 8 ICP iterations)	5462	248
Point cloud creation	1859	80

Figure 5: Time requirements of the individual parts of the reconstruction algorithm tested on the kitchen input set.

Figure 6: Reconstructed 3D model of the complex kitchen interior, based on 23 input frames. Resulting point clouds contains more than one million points and the reconstruction itself takes less than one minute to complete on the common today's hardware.

References

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

[2] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, September 1975.

[3] P.J. Besl and H.D. McKay. A method for registration of 3-D shapes. *Pattern Analysis and Machine Intelligence, IEEE Transactions*, 14(2):239–256, February 1992.

[4] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*, pages 73–90. McGraw-Hill Higher Education, 2nd edition, 2001.

[5] D. W. Eggert, A. Lorusso, and R. B. Fisher. Estimating 3-D rigid body transformations: a comparison of four major algorithms. *Machine Vision and Applications*, 9(5-6):272–290, March 1997.

[6] Christopher Evans. Notes on the OpenSURF library. Technical Report CSTR-09-001, University of Bristol, January 2009.

[7] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3):209–226, September 1977.

[8] D. Gossow, P. Decker, and D. Paulus. Robocup 2010. chapter An evaluation of open source SURF implementations, pages 169–179. Springer-Verlag, Berlin, Heidelberg, 2011.

[9] B. K. P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A*, 4(4):629–642, 1987.

[10] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, and A. Fitzgibbon. Kinectfusion: real-time 3D reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, UIST '11, pages 559–568, New York, USA, 2011. ACM.

[11] K. Kanatani. Analysis of 3-D rotation fitting. *IEEE Trans. Pattern Anal. Mach. Intell.*, 16(5):543–549, May 1994.

[12] D.T. Lee and C.K. Wong. Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees. *Acta Informatica*, 9:23–29, 1977.

[13] R. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon. KinectFusion: Real-time dense surface mapping and tracking. In *IEEE ISMAR*, pages 127–136, October 2011.

[14] Zhengyou Zhang. Iterative point matching for registration of free-form curves and surfaces. *International Journal of Computer Vision*, 13(2):119–152, October 1994.