KD-Tree Based Load Balancing for MMOG servers

CARLOS EDUARDO B. BEZERRA
JOÃO L. D. COMBA
CLÁUDIO F. R. GEYER
Universidade Federal do Rio Grande do Sul

MMOGs (massively multiplayer online games) are applications that require high bandwidth connections to work properly. This demand for bandwidth is specially critical on the servers that host the game. This happens because the typical number of simultaneous participants in this kind of game varies from a few hundreds to several tens of thousands, and the server is the one responsible for mediating the interaction between every pair of players connected to it. To deal with this problem, decentralized architectures with multiple servers have been proposed, where each server manages a region of the virtual environment of the game. Each player, then, connects only to the server that manages the region where he is playing. However, to distribute the load among the servers, it is necessary to devise an algorithm for partitioning the virtual environment. In order to readjust the load distribution during the game, this algorithm must be dynamic. Some work has already been made in this direction, but with a geometric algorithm, more appropriate than those found in the literature, it should be possible to reduce the distribution granularity without compromising the rebalancing time, or even reducing it. In this work, we propose the use of a kd-tree for dividing the virtual environment of the game into regions, each of which being designated to one of the servers. The split coordinates of the regions are adjusted dynamically according to the distribution of avatars in the virtual environment. We compared our algorithm to some approaches found in the literature and the simulation results show that our algorithm performed better in most aspects we analyzed.

Categories and Subject Descriptors: D.2.7 [**Software Engineering**]: Distribution and Maintenance—*documentation*; H.4.0 [**Information Systems Applications**]: General; I.7.2 [**Text Processing**]: Document Preparation—*languages*; *photocomposition*

General Terms: Documentation, Languages

Additional Key Words and Phrases: MMOGs, load balancing, distributed server, kd-trees

1. INTRODUCTION

The main characteristic of MMOGs is the large number of players interacting simultaneously, reaching the number of tens of thousands [**?**]. When using a client-server architecture for the players to communicate with one another, the server intermediates the communication between each pair of players.

To allow the interaction of players, each one of them sends his commands to the server, which calculates the resulting game state and sends it to all the players to whom the state change is relevant. We can see that the number of state update messages sent by the server may grow proportionally to the square of the number of players, if all players are interacting with one another. Obviously, depending on the number of players, the cost of maintaining a centralized infrastructure like this is too high, restricting the MMOG market to large companies with enough resources to pay the upkeep of the server.

In order to reduce this cost, several decentralized solutions have been proposed. Some of them use peer-to-peer networks, such as [**?**; **?**; **?**; **?**; **?**; **?**]. Others propose

Fig. 1. Division into cells and grouping into regions

the use of a distributed server composed of low-cost nodes connected through the Internet, as in [**?**; **?**; **?**; **?**]. Anyway, in all these approaches, the "world", or virtual environment of the game is divided into regions and for every region is assigned a server – or a group of peers to manage it, when using peer-to-peer networks. Each of these regions must have a content such that the load imposed on the corresponding server is not greater than its capacity.

When an *avatar* (representation of the player in the virtual environment) is located in a region, the player controlling that avatar connects to the server associated to that region. That server, then, is responsible for receiving the input from that player and for sending, in response, the update messages. When a server becomes overloaded due to an excessive number of avatars in its region and, therefore, more players to be updated, the division of the virtual environment must be recalculated in order to alleviate the overloaded server.

Usually, the virtual environment is divided into relatively small cells, which are then grouped into regions and distributed among the servers. However, this approach has a severe limitation in its granularity, since the cells have fixed size and position. Using a more appropriate geometric algorithm, it should be possible to achieve a better player distribution among different servers, making use of traditional techniques that are generally used for computer graphics.

In this work, we propose the utilization of a kd-tree to perform the partitioning of the virtual environment. When a server is overloaded, it triggers the load balancing, readjusting the limits of its region by changing the split coordinates stored in the kd-tree. A prototype has been developed and used in simulations. The results found in these simulations have been compared to previous results from approaches which use the cell division technique.

The text is organized as follows: in section 2, some related works are described; in section 3, the algorithm proposed here is presented in detail; in the sections 4 and 5, we present, respectively, the simulation details and its results and, in section 6, the conclusions of this work are presented.

2. RELATED WORK

Different authors have tried to address the problem of partitioning the virtual environment in MMOGs for distribution among multiple servers [**?**; **?**]. Generally, there is a static division into cells of fixed size and position. The cells are then grouped into regions (Figure 1), and each region is delegated to one of the servers. When one of them is overwhelmed, it seeks other servers, which can absorb part of the load. This is done by distributing one or more cells of the overloaded server to other servers.

[**?**], for example, propose a cell-oriented load balancing model. To balance the load, their algorithm finds, first, all clusters of cells that are managed by the overloaded server. The smallest cluster is selected and, from this cluster, it is chosen the cell which has the least interaction with other cells of the same server – the interaction between two cells A and B is defined by the authors as the number of pairs of avatars interacting with each other, one of them in A and the other one

Fig. 2. Graph representation of the virtual environment

Fig. 3. Space partitioning using a BSP tree

in B. The selected cell is then transferred to the least loaded server, considering "load" as the bandwidth used to send state updates to the players whose avatars are positioned in the cells managed by that server. This process is repeated until the server is no longer overloaded or there is no more servers capable of absorbing more load – in this case, one option could be to reduce the frequency at which state update messages are sent to the players, as suggested by [**?**].

In [**?**], it is also proposed the division into cells. To perform the division, the environment is represented by a graph (Figure 2), where each vertex represents a cell. Every edge in the graph connects two vertices representing neighboring cells. The weight of a vertex is the server's bandwidth occupied to send state updates to the players whose avatars are in the cell represented by that vertex. The interaction between any two cells define the weight of the edge connecting the corresponding vertices. To form the regions, the graph is partitioned using a greedy algorithm: starting from the heaviest vertex, at each step it is added the vertex connected by the heaviest edge to any of the vertices already selected, until the total weight of the partition of the graph – defined as the sum of the vertices' weights – reaches a certain threshold related to the total capacity of the server that will receive the region represented by that partition of the graph.

Although this approach works, there is a serious limitation on the distribution granularity it can achieve. If a finer granularity is desired, it is necessary to use very small cells, increasing the number of vertices in the graph that represents the virtual environment and, consequently, the time required to perform the balancing. Besides, the control message containing the list of cells designated to each server also becomes longer. Thus, it may be better to use another approach to perform the partitioning of the virtual environment, possibly using a more suitable data structure, such as the kd-tree [**?**].

This kind of data structure is generally used in computer graphics. However, as in MMOGs there is geometric information – such as the position of the avatars in the environment –, space partitioning trees can be used. Moreover, we cand find in the literature techniques for keeping the partitions defined by the tree with a similar "load". In [**?**], for example, it is sought to reduce the time needed to calculate the collisions between pairs of objects moving through space. The authors propose the use of a BSP (binary space partitioning) tree to distribute the objects in the scene (Figure 3). Obviously, if each object of a pair is completely inserted in a different partition, they do not collide and there is no need to perform a more complex test for this pair. Assuming an initial division, it is proposed by the authors a dynamic readjustment of the tree as objects move, balancing their distribution on the leaf-nodes of the tree and, therefore, minimizing the time required to perform the collision detection. Some of the ideas proposed by the authors may be used in the context of load balancing between servers in MMOGs.

Fig. 4. Balanced kd-trees built with the described algorithm

3. PROPOSED APPROACH

The load balancing approach proposed here is based on two criteria: first, the system should be considered heterogeneous (i.e. every server may have a different amount of resources) and, second, the load on each server is *not* proportional to the number of players connected to it, but to the amount of bandwidth required to send state update messages to them.

This choice is due to the fact that every player sends commands to the server at a constant rate, so the number of messages received by the server per unit time grows linearly with the number of players, whereas the number of state update messages sent by the server may be quadratic, in the worst case.

As mentioned in the introduction, to divide the environment of the game into regions, we propose the utilization of a data structure known as kd-tree. The vast majority of MMOGs, such as World of Warcraft [**?**], Ragnarok [**?**] and Lineage II [**?**], despite having three-dimensional graphics, the simulated world – cities, forests, swamps and points of interest in general – in these games is mapped in two dimensions. Therefore, we propose to use a kd-tree with k = 2.

Each node of the tree represents a region of the space and, moreover, in this node it is stored a split coordinate. Each one of the two children of that node represents a subdivision of the region represented by the parent node, and one of them represents the sub-region before the split coordinate and the other one, the sub-region containing points whose coordinates are greater than or equal to the split coordinate. The split axis (in the case of two dimensions, the axes x and y) of the coordinate stored alternates for every level of the tree – if the first level nodes store x-coordinates, the second level nodes store y-coordinates and so on. Every leaf node also represents a region of the space, but it does not store any split coordinate. Instead, it stores a list of the avatars present in that region. Finally, each leaf node is associated to a server of the game. When a server is overloaded, it triggers the load balancing, which uses the kd-tree to readjust the split coordinates that define its region, reducing the amount of content managed by it.

Every node of the tree also stores two other values: capacity and load of the subtree. The load of a non-leaf node is equal to the sum of the load of its children. Similarly, the capacity of a non-leaf node is equal to the sum of the capacity of its children nodes. For the leaf nodes, these values are the same of the server associated to each one of them. The tree root stores, therefore, the total weight of the game and the total capacity of the server system.

In the following sections, it will be described the construction of the tree, the calculation of the load associated with each server and the proposed balancing algorithm.

3.1 Building the kd-tree

To make an initial space division, it is constructed a balanced kd-tree. For this, we use the recursive function shown in Algorithm 1 to create the tree.

In Algorithm 1, the *id* value is used to calculate whether each node has children or not and, in the leaf nodes, it determines the server associated to the region

Algorithm 1 node::build_tree(id, level, num_servers)

```
if id + 2^level ≥ num_servers then
  left_child ← right_child ← NIL;
  return;
else
  left_child ← new_node();
  left_child.parent ← this;
  right_child ← new_node();
  right_child.parent ← this;
  left_child.build_tree(id, level + 1, num_servers);
  right_child.build_tree(id + 2^level, level + 1, num_servers);
end if
```

Fig. 5. A load splitting considering only the number os avatars

represented by each leaf of the tree. The purpose of this is to create a balanced tree, where the number of leaf nodes on each of the two sub-trees of any node differs, in the maximum, by one. In Figure 4 (a), we have a full kd-tree formed with this simple algorithm and, in Figure 4 (b), an incomplete kd-tree with six-leaf nodes. As we can see, every node of the tree in (b) has two sub-trees whose number of leaf nodes differs by one in the worst case.

3.2 Calculating the load of avatars and tree nodes

The definition of the split coordinate for every non-leaf node of the tree depends on how the avatars will be distributed among the regions. An initial idea might be to distribute the players among servers, so that the number of players on each server is proportional to the bandwidth of that server. To calculate the split coordinate, it would be enough to simply sort the avatars in an array along the axis used (x or y) by the tree node to split the space and, then, calculate the index in the vector, such that the number of elements before this index is proportional to the capacity of the left child and the number of elements from that index to the end of the array is proportional to the capacity of the right child (Figure 5). The complexity of this operation is $O(nlogn)$, due to the sorting of avatars.

However, this distribution is not optimal, for the load imposed by the players depends on how they are interacting with one another. For example, if the avatars of two players are distant from each other, there will be probably no interaction between them and, therefore, the server will need only to update every one of them about the outcome of his own actions – for these, the growth in the number of messages is linear with the number of players. On the other hand, if the avatars are close to each other, each player should be updated not only about the outcome of his own actions but also about the actions of every other player – in this case, the number of messages may grow quadratically with the number of players (Figure 6). For this reason, it is not sufficient only to consider the number of players to divide them among the servers.

A more appropriate way to divide the avatars is by considering the load imposed

Fig. 6. Relation between avatars and load

Fig. 7. Sweep of the sorted array of avatars

by each one of them on the server. A brute-force method for calculating the loads would be to get the distance separating each pair of avatars and, based on their interaction, calculate the number of messages that each player should receive by unit of time. This approach has complexity $O(n^2)$. However, if the avatars are sorted according to their coordinates on the axis used to divide the space in the kd-tree, this calculation may be performed in less time.

For this, two nested loops are used to sweep the avatars array, where each of the avatars contains a *load* variable initialized with zero. As the vector is sorted, the inner loop may start from an index before which it is known that no avatar a_j has relevance to that being referenced in the outer loop, a_i. It is used a variable *begin*, with initial value of zero: if the coordinate of a_j is smaller than that of a_i, with a difference greater than the maximum view range of the avatars, the variable *begin* is incremented. For every a_j which is at a distance smaller than the maximum view range, the *load* of a_i is increased according to the relevance of a_j to a_i. When the inner loop reaches an avatar a_j, such that its coordinate is greater than that of a_i, with a difference greater than the view range, the outer loop moves immediately to the next step, incrementing a_i and setting the value of a_j to that stored in *begin* (Figure 7).

Let *width* be the length of the virtual environment along the axis used for the splitting; let also *radius* be the maximum view range of the avatars, and n, the number of avatars. The number of relevance calculations, assuming that the avatars are uniformly distributed in the virtual environment is $O(m \times n)$, where m is the number of avatars compared in the internal loop, i.e. $m = \frac{2 \times radius \times n}{width}$. The complexity of sorting the avatars along one of the axes is $O(nlogn)$. Although it is still quadratic, the execution time is reduced significantly, depending on the size of the virtual environment and on the view range of the avatars. The algorithm could go further and sort each set of avatars a_j which are close (in one of the axes) to a_i according to the other axis and, again, perform a sweep eliminating those which are too far away, in both dimensions. The number of relevance calculations would be $O(p \times n)$, where p is the number of avatars close to a_i, considering the two axes of coordinates, i.e. $p = \frac{(2 \times radius)^2 \times n}{width \times height}$. In this case, *height* is the extension of the environment in the second axis taken as reference. Although there is a considerable reduction of the number of relevance calculations, it does not pay the time spent in sorting the sub-array of the avatars selected for each a_i. Adding up all the time spent on sort operations, it would be obtained a complexity of: $O(nlogn + n \times mlogm)$.

After calculating the load generated by each avatar, this value is used to define the load on each leaf node and, recursively, on the other nodes of the kd-tree. To each leaf node a server and a region of the virtual environment are assigned. The load of the leaf node is equal to the server's bandwidth used to send state updates to the players controlling the avatars located in its associated region. This way, the

Fig. 8. Avatar array sorted by x, containing a list sort by y

Fig. 9. Search for an ancestor node with enough resources

load of each leaf node is equal to the sum of the weights of the avatars located in the region represented by it.

3.3 Dynamic load balancing

Once the tree is built, each server is associated to a leaf node – which determines a region. All the state update messages to be sent to players whose avatars are located in a region must be sent by the corresponding server. When a server is overloaded, it may transfer part of the load assigned to it to some other server. To do this, the overloaded server collects some data from other servers and, using the kd-tree, it adjusts the split coordinates of the regions.

Every server maintains an array of the avatars located in the region managed by it, sorted according to the x coordinate. Also, each element of the array stores a pointer to another element, forming a chained list that is ordered according to the y coordinated of the avatars (Figure 8). By maintaining a local sorted avatar list on each server, the time required for balancing the load is somewhat reduced, for there will be no need for the server performing the rebalance to sort again the avatar lists sent by other servers. It will need only to merge all the avatars lists received from the other servers in an unique list, used to define the limits of the regions, what is done by changing the split coordinates which define the space partitions.

When the overloaded server initiates the rebalance, it runs an algorithm that traverses the kd-tree, beginning from the leaf node that defines its region and going one level up at each step until it finds an ancestor node with a capacity greater then or equal to the load. While this node is not found, the algorithm continues recursively up the tree until it reaches the root. For each node visited, a request for the information about all the avatars and the values of load and capacity is sent to the servers represented by the leaf nodes of the sub-tree to the left of that node (Figure 9). With these data, and its own list of avatars and values of load and capacity, the overloaded server can calculate the load and capacity of its ancestral node visited in the kd-tree, which are not known beforehand – these values are sent on-demand to save up some bandwidth of the servers and to keep the system scalable.

Reaching an ancestral node with capacity greater than or equal to the load – or the root of the tree, if no such node is found – the server that initiated the balance adjusts the split coordinates of the kd-tree nodes. For each node, it sets the split coordinate in a way such that the avatars are distributed according to the capacity of the node's children. For this, it is calculated the load fraction that should be assigned to each child node. The avatar list is then swept, stopping at the index i such that the total load of the avatars before i is approximately equal to the value defined as the load to be designated to the left child of the node whose split coordinate is being calculated (Figure 10). The children nodes have also, in turn, their split coordinates readjusted recursively, so that they are checked for validity

Fig. 10. Division of an avatar list between two brother nodes

– the split coordinate stored in a node must belong to the region defined by its ancestors in the kd-tree – and readjusted to follow the balance criteria defined.

As the avatar lists received from the other servers are already sorted along both axes, it is enough to merge these structures with the avatar list of the server which initiated the rebalance. Assuming that each server already calculated the weight of each avatar managed by it, the rebalance time is $O(nlogS)$, where n is the number of avatars in the game and S is the number of servers. The communication cost is $O(n)$, caused by the sending of data related to te n avatars. The merging of all avatar lists has $O(n)$ complexity, for the avatars were already sorted by the servers. At each level of the kd-tree, $O(n)$ avatars are swept in the worst case, in order to find the i index whose avatar's coordinate will be used to split the regions defined by each node of the tree (Figure 10). As this is a balanced tree with S leaf nodes, it has a height of $\lceil logS \rceil$.

4. SIMULATIONS

To evaluate the proposed dynamic load balancing algorithm, a virtual environment across which many avatars moved was simulated. Starting from a random point in the environment, each avatar moved according to the random waypoint model [**?**]. To force a load imbalance and stress the algorithms tested, we defined some *hotspots* – points of interest to which the avatars moved with a higher probability than to other parts of the map. This way, a higher concentration of avatars was formed in some areas. Although the movement model used is not very realistic in terms of the way the players move their avatars in real games, it was only used to verify the load balance algorithms simulated. For each algorithm tested, we simulated two situations: one with the presence of hotspots and one without hotspots.

The proposed approach was compared to the ones presented in section 2, from other authors. However, it is important to observe that the model employed by [**?**] considers hexagonal cells, while in our simulations we used rectangular cells. Furthermore, the authors considered that there is a transmission rate threshold, which is the same for all servers in the system. As we assume a heterogenous system, their algorithm was simulated considering that each server has its own transmission rate threshold, depending on the upload bandwidth available in each one of them. However, we kept what we consider the core idea of the authors' approach, which is the selection of the smallest cell cluster managed by the overload server, then choosing that cell with the lowest interaction with other cells of the same server, and finally the transferring of this cell to the least loaded server. Besides Ahmed's algorithm, we also simulated some of the ones proposed in [**?**] – Progrega and BFBCT.

The simulated virtual environment consisted of a two-dimensional space, with 750 moving avatars, whose players were divided among eight servers $(S_1, S_2, ..., S_8)$, each of which related to one of the regions determined by the balancing algorithm. For the cell-oriented approaches simulated, the space was divided into a 15×15 cell grid, or 225 cells. The capacity of each server S_i was equal to $i \times 20000$, forming a heterogeneous system. This heterogeneity allowed us to evaluate the

Fig. 11. Average load on each server (by algorithm, without hotspots)

Fig. 12. Average load on each server (by algorithm, with hotspots)

Fig. 13. Average deviation of the ideal balance of the servers (without hotspots)

Fig. 14. Average deviation of the ideal balance of the servers (with hotspots)

load balancing algorithms simulated according to the criterion of proportionality of the load distribution on the servers.

In addition to evaluate the algorithms according to the proportionality of the load distribution, it was also considered the number of player migrations between servers. Each migration involves a player connecting to the new server and disconnecting from the old one. This kind of situation may occur in two cases: the avatar moved, changing the region in which it is located and, consequently, changing the server to which its player is connected; or the avatar was not moving and still its player had to migrate to a new server. In the latter case, obviously the player's transfer was due to a rebalancing. An ideal balancing algorithm performs the load redistribution requiring the minimum possible number of player transfers between servers, while keeping the load on each server proportional to its capacity.

Finally, the inter-server communication overhead will also be evaluated. It occurs when two players are interacting, but each one of them is connected to a different server. Although the algorithm proposed in this work does not address this problem directly, it would be interesting to evaluate how the load distribution performed by it influences the communication between the servers.

5. RESULTS

Figures 11 and 12 present the average load (plus the inter-server communication overhead) on each server, for each algorithm tested. The first figure shows the values in a situation without hotspots and, therefore, a smaller total load. The second, in turn, presents the load distribution when the server system is overloaded. We can see, in Figure 11, that all algorithms have met the objective of keeping the load on each server less than or equal to its capacity, when the system has sufficient resources to do so. In Figure 12, it is demonstrated that all the algorithms managed to dilute – in a more or less proportional manner – the load excess on the servers. It is important to observe, however, that the load shown in Figure 12 is only theoretical. Each server will perform some kind of "graceful degradation" in order to keep the load under its capacity. For example, the update frequency might be reduced and access to the game could be denied for new players attempting to join, which is a common practice in most MMOGs.

In figures 13 and 14, it is shown how much the balance generated by each algorithm deviates from an ideal balance – that is, how much, on the average, the load on the servers deviate from a value exactly proportional to the capacity of each one of them – over time. It is possible to observe that, in both situations – with

Fig. 15. Player migrations between servers (without hotspots)

Fig. 16. Player migrations between servers (with hotspots)

Fig. 17. Inter-server communication for each algorithm over time (without hotspots)

Fig. 18. Inter-server communication for each algorithm over time (with hotspots)

and without hotspots – the algorithm that uses the kd-tree has the least deviation. This is the due to the fine granularity of its distribution, which, unlike the other approaches tested, is not limited by the size of a cell. In the situation with hotspots, the algorithm that uses the kd-tree is particularly effective, because rebalance is needed. In a situation where the system has more resources than necessary, the proportionality of the distribution is not as important: it is enough that each server manages a load smaller than its capacity.

Regarding player migrations between servers, all the algorithms – except BFBCT – had a similar number of user migrations in the absence of hotspots (Figure 15). This happens because the load of the game is less than the total capacity of the server system, which required less rebalancing and, thus, caused less migrations of players between servers. Figure 16, however, demonstrates that the algorithm that uses the kd-tree had a significantly lower number of user migrations than the other approaches. This is due, in the first place, to the fact that the regions defined by the leaf nodes of the kd-tree are necessarily contiguous, and each server was linked to only one leaf node. An avatar moving across the environment divided into very fragmented regions constantly crosses the borders between these regions and causes, therefore, its player to migrate from server to server repeatedly. Another reason for this result is that each rebalancing executed with the kd-tree gets much closer to an ideal distribution than the cell-based algorithms – again, thanks to the finer granularity of the kd-tree based distribution –, requiring less future rebalancing and, thus, causing less player migrations.

Finally, it is shown the amount of communication between servers for each simulated algorithm, over time. In Figure 17, all algorithms have similar results, and the one which uses the kd-tree is slightly better than the others. This is also explained by the fact that the regions are contiguous, minimizing the number of boundaries between them and, consequently, reducing the probability of occurring interactions between pairs of avatars, each one in a different region. In Figure 18, it is possible to see that the inter-server communication caused by Progrega was considerably lower than all the others in a situation of system overload. The reason for this is that its main goal – besides balancing the load – is precisely to reduce the communication between servers. However, even not considering the additional cost, the algorithm that uses the kd-tree got second place in this criterion.

6. CONCLUSIONS

In this work, we proposed the use of a kd-tree to partition the virtual environment of MMOGs and perform the load balancing of servers by recursively adjusting the split coordinates stored in its nodes. One of the conclusions reached was that the use of kd-trees to make this partitioning allows a fine granularity of the load distribution, while the readjustment of the regions becomes simpler – by recursively traversing the tree – than the common approaches, based on cells and/or graph partitioning.

The finer granularity allows for a better balancing, so that the load assigned to each server is close to the ideal value that should be assigned to it. This better balance also helped to reduce the number of migrations, by performing less rebalancing operations. The fact that the regions defined by the kd-tree are necessarily contiguous was one of the factors that contributed to the results of the proposed algorithm, which was better than the other algorithms simulated in most of the criteria considered.

In conclusion, it was possible to use methods that can reduce the complexity of each rebalancing operation. This is due, first, to the reduction of the number of operations for calculating the relevance between pairs of avatars by sweeping a sorted avatar list and, secondly, to keeping at each server an avatar list already sorted in both dimensions, saving the time that would be spent on sorting the avatars when they were received by the server executing the rebalance.

7. THE TITLE PAGE

7.1 The Title, Author(s), and Abstract

Following order is mandatory to generate a correct title page:[1]

```
\documentclass{acmtrans2m}
            %\acmVolume{V}
            %\acmNumber{N}
            %\acmYear{YY}
            %\acmMonth{Month}
            \markboth{}{}
            \title{}
            \author{}
            \begin{abstract}
            ...
            \end{abstract}
            \category{}{}{}
            \terms{}
            \keywords{}
            \begin{document}
            \begin{bottomstuff}
            ...
            \end{bottomstuff}
```

[1]Section 7 has been rewritten by Marco Aiello (email: aiellom@acm.org) to explain the new structure of the class file as of 2001/06/01. The major novelty is the introduction of options for the different transactions and the the automatic generation of footers and permission statements.

```
\maketitle
```

The `\documentclass[journalName]{acmtrans2m}` takes as option the specific transaction one is preparing. The transactions currently supported are:

option name	journal
`acmjacm`	Journal of the ACM
`acmtocl`	Transactions on Computational Logic
`acmtodaes`	Transactions on Design Automation of Electronic Systems
`acmtods`	Transactions on Database Systems
`acmtogs`	Transactions on Graphics
`acmtoms`	Transactions on Mathematical Software
`acmtoplas`	Transactions on Programming Languages and Systems

For example, to prepare a manuscript for the Transactions on Compuational Logic the file should begin with

```
\documentclass[acmtocl]{acmtrans2m}
```

The four commands

```
%\acmVolume{V}
%\acmNumber{N}
%\acmYear{YY}
%\acmMonth{Month}
```

are needed to generate footer and copyright information. The commands store the following information: volume number, issue number, last two digits of the year of publication, and month name in English, respectively. The appropriate values will be communicated by the Editor-in-Chief upon acceptance of the final version of the paper.

7.1.1 *Title and Author.* The LaTeX `\title` and `\author` declarations and the `\maketitle` command are employed as usual. However, the user must format the author a little differently to match the ACM standard. The following example [**?**] illustrates most features:

```
\author{JAMES E. ARCHER, JR.\\ Rational Machines
        \and RICHARD CONWAY and FRED B. SCHNEIDER \\
             Cornell University}
```

Note that authors' names are in uppercase letters, authors are separated from their affiliation by a `\\` command, multiple authors with the same affiliation are separated by "and" (or commas and "and" if there are more than two), and authors with different affiliations are separated by an `\and` command. The following example [**?**] shows what to do if there are more than two affiliations:

```
\author{E. KORACH \\ IBM Israel \\
        D. ROTEM \\ University of Waterloo
        \and N. SANTORO \\ Carleton University}
```

In both the title and the author, you may have to insert `\\` commands if lines need to be broken.

7.1.2 *Abstract.* The abstract is typed as usual with the `abstract` environment. However, this environment must come before the `\maketitle` command.

7.2 Content Indicators and Keywords

The content indicators and keywords are entered with LaTeX declarations. The CR categories are indicated with `\category` declarations. The first CR category of this article, appearing right below the abstract, was entered with the following command:

```
\category{D.2.7}{Software Engineering}{Distribution and
  Maintenance}[Documentation]
```

Note that the last argument (which contains the subject descriptors) is optional, since some categories have none. Multiple subject descriptors are separated by `\and` commands, as in the last category of this article:

```
\category{I.7.2}{Text Processing}{Document Preparation}
         [Languages \and Photocomposition]
```

Use a separate `\category` declaration for each CR category; they will be listed in the order that the commands appear. The `\category` commands must precede the `\maketitle` command.

The General Terms are declared with a (single) `\terms` command as in the one for this article:

```
\terms{Documentation, Languages}
```

The `\terms` declaration must come before the `\maketitle` command. The terms *must* be chosen from the following list:

> Algorithms; Design; Documentation; Economics; Experimentation; Human factors; Languages; Legal aspects; Management; Measurement; Performance; Reliability; Security; Standardization; Theory; Verification;

The general terms are orthogonal to the Categories, at least theoretically, and so may be applied to any elements of the classification tree.

Think of them as 'perspectives' from which any topic may be approached. Thus you could use *Theory* or *Performance* for an article about *C.2.1 Distributed Networks*. However, some of these general terms actually slide over into content areas. Thus *Legal aspects* is a general term applicable to any category, but also an entire node in the tree, *K.5*, devoted to *Legal aspects of computing,* with many sub-topics.

So, though perhaps not perfect, the General Terms are most useful in online searches when used in combination with categories.

The "Additional Keywords and Phrases" item on the title page is provided by the `\keywords` declaration, **listed alphabetically**. For this article, they were produced by the following command:

```
\keywords{Document preparation, publications, typesetting}
```

There is no prescribed list of "additional keywords;" use any that you want.

7.3 The Bottom of the Title Page

The bottom of the article's title page contains acknowledgment of support, the author(s) address(es), a "permission to copy" statement, and a line containing a copyright symbol (©) and a mysterious number. This is all entered with a `bottomstuff` environment; there must be no blank line after the `\begin{bottomstuff}` command.

7.4 The Page Headers

`\markboth{}{}` generates the left- and right-page headers. The first argument is the author's name(s):

—If there is one author, then use author's full name (ex. Leslie Lamport);
—If there are two authors, then abbreviate each author's first name (L. Lamport and A. Appel);
—If there are more than two authors, then the format is Leslie Lamport et al.

The second argument of `markboth` is the title; if the title is too long, contract it by omitting subtitles and phrases, not by abbreviating words.

8. ORDINARY TEXT

Most of the body of the text is typed just as in an ordinary document. This section lists the differences.

8.1 Lists

8.1.1 *Enumeration and Itemization.* Let's begin with enumeration.

(1) The ACM style has two different formats for itemized lists, which I will call the *long* and *short* formats. The long format is generally used when the individual items are more than two or three lines long, but ACM has been inconsistent in their choice of format, sometimes using the long format for lists whose items are all one or two lines long and the short format for lists of long items. This list is an example of the long format.

(2) The ordinary `enumerate` environment produces the short format. For the long format, use the `longenum` environment.

(a) This inner enumeration uses the short format.
(b) It was produced using LaTeX's ordinary `enumerate` environment.
(c) ACM has no standard for enumerations nested more than two levels deep, so the `acmtrans` style does not handle them well.

Itemized lists are similar to enumerated ones.

— As with enumerations, there is a long and a short format for itemized lists. This list is in the long format.

— The long format is produced by the `longitem` environment. The ordinary `itemize` environment uses the short format.

—This is an itemized list using the short format.
—It was produced with the `itemize` environment that is used in ordinary LaTeX input.

It is interesting to observe that the style of tick mark used for an itemization changed around 1985 from an en dash (–) to an em dash (—).

8.1.2 *Descriptions.* A list is a sequence of displayed text elements, called items, each composed of the following two elements:

label: A marker that identifies or sets off the item. It is a number in an enumerated list and a tick mark in an itemized list.

item body: The text of the item. It is usually ordinary prose, but sometimes consists of an equation, a program statement, etc.

Or another paragraph, which will be indented like normal paragraphs.

When the labels of a list are names rather than numbers or tick marks, the list is called a *description* list. The ACM style has both long and short description lists. The above list is a short description list; the bodies of all the items are indented enough to accommodate the widest label. The following list is a long description list. The `acmtrans` style provides both kinds of description lists:

short. The `describe` environment takes an argument, which should be the same as the argument of the `\item` command that produces the widest label. Thus, the above description list was begun with the command

```
\begin{describe}{{\em item body\/}:}
```

A description label is often emphasized in some way; in this example I used the LaTeX `\em` command, italicized the label. The ACM appears to have no standard convention for formatting the labels of a description list, so the `describe` environment leaves the label formatting up to you. An `\hfill` command can be used to produce a label like "*gnu* –" where *gnu* is flush left against the margin and the "–" is aligned flush right next to the item body.

long. The standard LaTeX `description` environment produces a long description list. It italicizes the labels, and puts a period after them, which seems to be what is done in the ACM transactions.

8.2 Theorems, Etc.

LaTeX provides a single class of theorem-like environments, which are defined with the `\newtheorem` command. The ACM transactions style divides this class into two subclasses that are formatted differently. The first class includes theorems, corollaries, lemmas, and propositions. It is produced with the `\newtheorem` command. Such a theorem-like environment is often followed by a proof, for which the `acmtrans` style provides a `proof` environment.

THEOREM 8.2.1. *Notice that theorems are numbered inside the nearest section subsection.*

When listing within the theorem environment, this style will now produce roman parantheses. Thank you David Sands.

PROOF. This theorem is an instance of `subtheorem`, theorems nested in subsections. □

Please use this set of definitions, possibly extended by your additional `\newtheorem` items:

```
\newtheorem{theorem}{Theorem}[section]
\newtheorem{conjecture}[theorem]{Conjecture}
\newtheorem{corollary}[theorem]{Corollary}
\newtheorem{proposition}[theorem]{Proposition}
\newtheorem{lemma}[theorem]{Lemma}
\newdef{definition}[theorem]{Definition}
\newdef{remark}[theorem]{Remark}
```

The second subclass of theorem-like environments includes ones for definitions, examples, and remarks. These environments are defined with the `\newdef` command, (used just above) which works the same as `\newtheorem`. Here is an example of such an environment.

Definition 8.2.2. This is an example of a Definition, typed with an `subexample` environment defined with `\newdef`. As you can see theorems are italicized and definitions are not.

Sometimes theorem-like environments are numbered in unusual ways, or are identified by a name. Consider the following example from **?**].

PROPERTY 8.2.3 Ca. *Let* $syn \in Syn$, $occ \in Occ$ *be maximal and* $sta \in Sta$. *Then* $Tcol[[syn]]\ occ\ sta{\downarrow}1 = Tsto[[syn]]\ sta$.

PROOF OF PROPERTY Ca. By straightforward structural induction, and is omitted. □

It was obtained by giving optional arguments to the `property` environment (defined with `\newtheorem`) and the `proof` environment and was typed as follows.

```
\begin{subproperty}[{\rm Ca}] Let ...  \end{subproperty}
\begin{proof}[of Property {\rm Ca}]  By straightforward ...
```

Notice that the optional argument to the `property` environment suppresses the automatic numbering. If a null optional argument were given to this environment by typing "`[]`", then it would have produced the label "PROPERTY." This is how unnumbered theorems, etc. are produced.

8.3 Overfull hbox - Stretching/filling one horizontal line

To solve a line break due to "Overfull `\hbox`", here is a plain TEX solution; here `\hsize` is the default setting of acmtrans.sty:

```
\hbox to \hsize{line sentence to be stretched}
```

This can be used in a list environment as well but `\hsize` declared to a reduce dimension:

```
\hbox{\vbox{\hsize = less than the default setting
\hbox to \hsize{line sentence to be stretched}}}
```

8.4 Programs

Good formatting of programs requires a knowledge of their semantics, and is beyond the scope of a document production system. While "pretty printers" are useful for

```
type date =
  record day: 1..31;
         month: 1..12;
         year: integer
  end
var mybirth, today : date;
var myage : integer;
```

Fig. 19. An example of a program centered in a figure

handling the many pages of a real program, the short examples that are published in articles should be formatted by hand to improve their clarity. The LaTeX `tabbing` environment makes the formatting of programs relatively easy, especially if the user defines commands for his particular language constructs. One may also use the `verbatim` environment.

The ACM transactions style requires that programs be formatted with different size fonts, depending upon whether they appear in the text or in a figure, but that is handled by the figure macro which automatically sets the correct font size. Moreover, programs in running text should be indented two ems on each side (as provided by the `quote` environment), and programs in regular figures should be centered. (Programs in "narrow figures" (q.v.) are left or right justified automatically).

Here is an example of a program:

```
type date =
  record day: 1..31;
         month: 1..12;
         year: integer
  end
var mybirth, today : date;
var myage : integer;
```

Figure 19 shows how the same program looks in a figure.

In addition to formatting programs, the `tabbing` environment may be used for similar displayed material such as BNF syntax specifications and rewrite rules.

8.5 User-specified Formatting

If LaTeX does not provide a particular text structure, the user must define it himself and specify how it is to be formatted. This is most easily done by defining the new structure in terms of existing ones; the LaTeX `list` and `trivlist` environments are useful tools. However, it is occasionally necessary for the user to provide explicit formatting commands.

The best guide to how something should be formatted is what has been done in the ACM transactions. While horizontal spacing tends to depend strongly upon the particular text, there is a standard amount of vertical space used to set off text. The ordinary LaTeX `\medskip` command produces a vertical space of the appropriate size.

	$\perp$	F	T
$\perp$	$\perp$	$\perp$	T
F	$\perp$	F	T
T	$\perp$	T	T

Fig. 20. The truth table for the parallel-or.

```
type date =
  record day: 1..31;
         month: 1..12;
         year: integer
  end
var mybirth, today : date;
var myage : integer;
```

Fig. 21. An example of a program displayed in a figure.

9. FIGURES AND TABLES

9.1 Figures

The ordinary LaTeX `figure` environment works as usual. Figure 20, which is Figure 6 of **?**], a bogus reference, was produced in this way. Note that figures should never appear in the text or at the bottom of a page. (If you use the figure placement optional argument, use only `t` or `p` or both; do not use `h` or `b`).

Some figures (and tables) have no caption except for the figure number. For such figures (and tables), one uses a `\nocaption` command, which has no argument, instead of the `\caption` command.

In addition to this method of formatting figures, the ACM transactions also uses figures with side captions, as in Figure 21. Such a figure is produced with the `narrowfig` environment. This environment has a single mandatory argument, which is the width of the figure. Note that if the figure is generated by `tabbing` or `tabular`, one can safely overestimate the size. It works just like the ordinary `figure` environment, except it must contain only one `\caption` or `\nocaption` command, which must come after the figure itself.

The `narrowfig` environment should obviously not be used unless the figure is narrow enough to leave a reasonable amount of space beside it for the caption. The ACM seems to have no consistent policy for choosing which style of figure to employ.

9.2 Tables

The ordinary LaTeX `table` environment can be used, but it requires the user to add formatting commands to match the ACM transactions style. This formatting is performed automatically if the `acmtable` environment is used instead, producing the result shown in Table I, which shows the same table displayed in Figure 20. This environment has a mandatory argument that equals the width of the table—more precisely, it specifies the width of the rules above and below the table. There must be only one `\caption` or `\nocaption` command, which must come after the text of the table. (Even though the table caption is printed above the table, the `\caption` command comes after the table in the input file.)

Table I. The truth table for the parallel-or.

	$\perp$	F	T
$\perp$	$\perp$	$\perp$	T
F	$\perp$	F	T
T	$\perp$	T	T

10. THE END OF THE DOCUMENT

10.1 Appendix

The appendix (if the article has one) should precede the acknowledgments (if any) and bibliography. If the appendix isn't broken into separate sections, then you should add the following commands after the `\appendix` command:

```
\section*{APPENDIX}
\setcounter{section}{1}
```

Setting the counter is necessary so that numbered subsections and theorems will have the names "A.N" in the text.

For an article with multiple appendices, one begins the appendix with an `\appendix` followed by `\section*{APPENDIX}`, and then starts each appendix with an ordinary `\section` command.

Information about electronic appendices is given in Section 13 and in the Appendix.

10.2 Acknowledgments

An optional acknowledgments section follows all the text of the article, including any appendices. It is produced with the `acks` environment. (Since I can never remember how many *e*'s there are in *acknowledgments*, it seemed like an abbreviation was in order for the environment name. One may also spell out the name as `acknowledgments`). Sometimes, there is just a single acknowledgment. This may be given using the `ack` or `acknowledgment` environment.

10.3 Bibliography

The bibliography follows the acknowledgments, and is the last significant body of text in the article. It is produced by the usual LaTeX commands.

In 1993 the ACM changed bibliography styles, from numerical citations [13] to author's name and year [**?**]. The user is encouraged to let LaTeX produce his bibliography with the `\bibliography` command, letting BibTeX handle the formatting of the entries. The `acmtrans.sty` bibliography style file now generates citations in this format. Put

```
\bibliographystyle{acmtrans}
```

between the `\begin{document}` and the `\end{document}`.

The conventional `\cite` command will generate citations as usual in LaTeX. Note that the style file automatically omits repeating author names [**?**; **?**]. If you mention the work explicitly in your prose, you should use `\citeN` command. This command generates for example, **?**] discusses denotational program transformations. Or, you use `\citeyear` and say that Nielson [**?**] discusses them. The

command `\shortcite` is an alias for `\citeyear`. Either command may be used in cases where one refers to multiple works (of the same authors!). For example, `Nielson~\shortcite{7:3:359,test}` generates Nielson [**?**; **?**].

More variations of `\cite` are discussed in comments in the `acmtrans.sty` file. Here are some **examples** on how to get

(1) Appel [1996] → using either `\citeN` or `\citeyear`

(2) [Kempe 1879] → `\cite{kempe79}`

(3) Appel [1995; 1996] → `\shortcite{ref1-key,ref2-key}`

(4) Filé [1981a; 1981b] → `Fil\'{e}~\shortcite{engelfriet/file:81sweep, engelfriet/file:81passes}` or simply `\shortcite{ref1-key, ref2-key}`

(5) [Appel and Shao 1992; Shao and Appel 1994] → `\cite{appel-zhong-lsc92, shao94:clo}`

(6) Chow and Harrison [1992; 1994] → Chow and Harrison [`\citeyearNP{CH-popl92}; \citeyearNP{CH-iccl94}`]

(7) [Chow and Harrison 1992; 1994; Cousot and Cousot 1984] → `[\citeNP{CH-popl92}; \citeyearNP{CH-iccl94}; \citeNP{CC-apct77}]`

(8) [Cytron et al. 1991] → `\cite{cytron-et-al-toplas91}`

(9) Briggs et al. [1994] → `\citeN{briggs-cooper-torczon-toplas94}` or

(10) Duri et al. [1993] → `Duri~et~al.~\citeyear{DBDS-sigsoft93}`

(11) [Chaitin 1982; Chaitin et al. 1981] → `\cite{chaitin-pldi82,chaitin-et-al-cl81}`

(12) [Alblas 1991; Deransart et al. 1988; Knuth 1868] → `\cite{alblas:91intro, deransart/jourdan/lorho:88ag,knuth:68semantics}`

(13) [Gary and Johnson 1979] → `\cite{garey-johnson-bk79}`

(14) [Brand and Zafiropulo 1983; Gouda et al. 1984;1987] → `[\citeNP{brand83}; \citeNP{gouda84};\citeyearNP{gouda87}]`

The list will be updated as we find unique cases.

10.4 Received Date

The article should end by the following lines:

```
\begin{received}
Received Month Year;
revised Month Year; accepted Month Year
\end{received}
\end{document}
```

The actual dates will be supplied by the Editor-in-Chief. The `\endreceived` command activates the `@lastpg` label. Omitting the environment will result in a undefined label warning.

11. RUNNING HEADS AND FEET

The running foot of all but the title page of the article is declared with the `\runningfoot` command. It contains the name of the journal, volume, number, and date. The foot for the title page contains this information plus the page numbers. It is declared with the `\firstfoot` command.

The `\pages` command prints the page numbers of the article, producing something like "123–132". It is implemented with the LaTeX `\pageref` command, so it will not produce the correct page numbers the first time the file is run through LaTeX, or if the number of the first or last page has changed since the last time.

The default page style for the `acmtrans` style is `myheadings`. Thus, a `\markboth` command is used to set the running heads. The left head contains the author's name (or authors' names) and the right head contains the title. For long titles, some contraction of the title is used.

At present, the ACM prefers to strip in their own running heads and feet, so it is unnecessary to worry about them when producing camera-ready copy.

12. INTERACTING WITH ACM'S PRODUCTION STAFF

12.1 ACM Copy Editors' Preferences

We wish to thank the authors who have mailed to us copies of their page proofs. From the page proofs (and explicit requests from ACM), here is a list of items most frequently marked up by the copy editors.

Center. Displayed items like figures, pieces of program codes, and equations. Equation numbers to appear right flushed. Remember to add a period if the displayed equation ends in a sentence.

Fonts. Following items in regular Roman font:
—such words like et al., e.g., i.e., ad hoc, and bona fide, and
—the body of the definition environment.

Etc. No double period when a sentence ends with "etc."

Figure. Spell out the word "figure" when using in a sentence, as in Figure 1. Use a period as in "Fig. 1." and not a colon as in "Fig. 1:"

Contractions. Avoid the use of contractions; for example, use *do not* instead of *don't*. (If you prefer to use contractions to avoid stuffiness, you'll have to tell the copy editor explicitly after you get the marked-up proofs.)

Conference References. Please use *4th International Conf. on Document Formatting*, not *Fourth International Conference ...*; that is, do not spell out the number.

Journal References. Figure 22 gives common journal abbreviations; there is a duplicate list in the acmtrans.bst file for your convenience. Using the *abbreviation* feature of BibTeX for journal names (and months!) makes it easy to follow the rules.

13. ELECTRONIC APPENDICES

Because of severe constraints on how many pages it can print, some ACM journals accept some articles with *electronic appendices:* appendices in Postscript format that will not appear in the printed article but will be available separately. If your article is accepted with an electronic appendix, you should put an appendix header where the appendix normally belongs (before the "acknowledgments"). The body of the electronic appendix should be given after the references. The `appendixhead` command is given

acmcs	ACM Comput. Surv.	jlp	J. Logic Program.
acmlett	ACM Lett. Program. Lang. Syst.	jcss	J. Comput. Syst. Sci.
acta	Acta Inf.	jsmrp	J. Softw. Maint. Res. Pract.
al	Ada Lett.	jss	J. Syst. Softw.
acr	Adv. Comput. Res.	jlsc	J. Lisp Symb. Comput.
bit	Bit	lpls	Lett. Program. Lang. Syst.
cacm	Commun. ACM	mscs	Math. Struct. Comput. Sci.
cj	Comput. J.	mst	Math. Syst. Theor.
cn	Comput. J.	ngc	New Gen. Comput.
cl	Comput. Lang.	scp	Sci. Comput. Program.
ict	Inf. Contr.	sicomp	SIAM J. Comput.
ieebcs	IEE/BCS Softw. Eng. J.	spe	Softw. Pract. Exper.
ieees	IEEE Softw.	tocs	ACM Trans. Comput. Syst.
ieeese	IEEE Trans. Softw. Eng.	tods	ACM Trans. Database Syst.
ieeetc	IEEE Trans. Comput.	tog	ACM Trans. Graphics
ieeetpds	IEEE Trans. Parall. Distrib. Syst.	toms	ACM Trans. Math. Softw.
ieeetit	IEEE Trans. Inf. Theory	toois	ACM Trans. Office Inf. Syst.
ipl	Inf. Process. Lett.	toplas	ACM Trans. Program. Lang. Syst.
icp	Inf. Comput.	tcs	Theor. Comput. Sci.
ist	Inf. Softw. Tech.	tr	Tech. Rep.
ijsa	Int. J. Supercomput. Appl.	jf	J. Funct. Program.
ijpp	Int. J. Parallel Program.	jlc	J. Logic and Comput.

Fig. 22. Common journal abbreviations.

`\appendixhead{`*URLend*`}`

where *URLend* will be determined by the Editor (it is usually the last name of the first author).

In case your paper will have an electronic appendix, the part of the paper that will appear in print should LaTeX correctly, i.e. in this part no LaTeX references (`\ref`) should be made to the electronic appendix.

The result of `\appendixhead` looks like this:[2]

ELECTRONIC APPENDIX

The electronic appendix for this article can be accessed in the ACM Digital Library by visiting the following URL: `http://www.acm.org/pubs/citations/journals/tocl/2001-2-3/p111-URLend`. Of course, the latter URL is just an example and it corresponds to no actual electronic appendix.

ACKNOWLEDGMENTS

We wish to thank Howard Trickey for providing the now-obsolete version of the `acmtrans` bibliography style, and for giving advice on creating the new one; Rebecca Davies for helping to adapt Glenn Paulley's "Chicago" bibliography style to fit the new ACM style; and Marilyn Salmansohn and George Criscione for providing information on the official ACM transactions style.

Received February 1986; November 1993; accepted January 1996

[2]See the end of this document for the remainder of the explanation of electronic appendices

The contents of the electronic appendix is written after the references and the "received" environment. The electronic appendix is started by an `\elecappendix` command:

```
\elecappendix
```

A. SPLITTING OFF THE ELECTRONIC APPENDIX

If you have an electronic appendix, only the main body of the article, up through and including the description of how to obtain the electronic appendix, will be printed in the journal.

It will be necessary to split your dvi or Postscript file into two parts: one to be printed, the other to be available by FTP. Please split your Postscript into two separate postscript files using `dvipages`, `pslpr` or `psselect` and email them separately to the Editor.

Note that the pages of the appendix are numbered App-1, App-2, etc. so as not to interfere with the normal journal pagination.

B. SINGLE APPENDIX

When an article has a single electronic appendix, then after the `\elecappendix` command, type the following.

```
\setcounter{section}{1}
```

If the text starts immediately, add a `\medskip` to set off the text from the horizontal rule created by `\elecappendix`.

C. MULTIPLE APPENDICES

For an article with multiple electronic appendices, one begins the appendix with an `\elecappendix` command, then starts each appendix with an ordinary `\section` command. Lower levels of sectioning are produced by the ordinary sectioning commands.