# GRAPHICAL MODELING OF RECURSION

**Rodica Baciu & Dorin Sima**
"Lucian Blaga" University of Sibiu, "Hermann Oberth" Faculty of Engineering
Department of Computer Science & Automatic Control
2400, Sibiu, Str. Emil Cioran, nr.4
E-mail: rodica.baciu@ulbsibiu.ro
E-mail: dorin.sima@ulbsibiu.ro

**Abstract:** Recursion is a fundamental concept in computer science. Creating a correct mental model for the concept of recursion allows the understanding of recursive algorithms and their correct implementation. Teaching recursion is not an easy task for teachers. Generally, supporting abstract concepts through graphics leads to a better understanding of them. This paper presents a graphical application that allows the use of graphics in order to explain the concepts of recursion and trees.

***Key Words:*** recursion, octree, CAL, modeling, solid representation

## INTRODUCTION

Recursion is a central concept in computer science, both for theoreticians and users. The concept of recursion places a central part in compiler cunstruction, data structures, artificial intelligence, problem solving, language theory, database, graphics, operating systems and many other fields. Recursive algorithms often provide smarter solutions to complex problems than the iterative ones, especially when combined with recursive data structures. The teaching experience of various professors has proved that teaching recursion is not an easy task. Specialized studies [Götschi 2003] have proved that a correct mental model of recursion allows a correct implementation of recursive algorithms.

There are a lot of research studies on modeling and visualization of recursion [Augenstein 1976; George 2000; Haberman 2002; Henderson 1989; Naps 1989; Stern 2002]. For example Thomas L. Naps [Naps 1989] shows how he built a class designed "to enable students to annotate their Java code with visual slides that depict the state of the algorithms they are coding."

Linda Stern and Lee Naish [Stern 2002] show how they have built an application for learning programming in which "the execution of the algorithm is traced with a cursor that progresses through the pseudocode, while the animation displays a conceptual representation of the steps in the pseudocode." They conclude "the animation helps students to grasp the overall concept of an algorithm, while the more detailed views help them understand the workings of the algorithm at a procedural level."

Carlisle E. George [George 2000] has implemented a tutorial, which "uses visualization and animation as well as sound and color to simulate the dynamic-logical model of subprogram calls."

The application that we present is useful in forming the mental concept of recursion. The verbal explanations and the examples given for understanding the process of recursion don't have the force of a graphical interactive presentation. This year one of the difficulties encountered in the final examination on the subject "Programming in assembler language" was the implementation of a recursive algorithm. Statistics have shown that only 24% students implemented correctly the algorithm and 59% haven't even tried to solve the problem. All the students considered that the most difficult task was the implementation of recursion.

## OCTREES

The basis of application for recursion modeling is the representation of solids through octrees. Octrees are solid models that are generated by a recursive subdivision of a finite cubic universe and that represent the tree of the subdivision process (Figure 1). The root of the octree represents the universe. It is a cube of edge length 2", called the maximum scale. The scale of an octree node is defined as the length of the edges of the corresponding cube. The root cube is divided into eight identical octants of scale $2^{n-1}$, which are represented by the tree nodes pointed at by the root at the second level of the tree. Valid terminal nodes include *white* nodes (completely outside the solid) and *black* nodes (completely inside the solid) (Figure 2). Whenever an octant cannot be represented as a valid terminal node, it is denoted as a *gray* node and is divided into eight other identical cubes, which are represented as descendants of the octant in question. This process is repeated recursively until either terminal nodes or cubes of

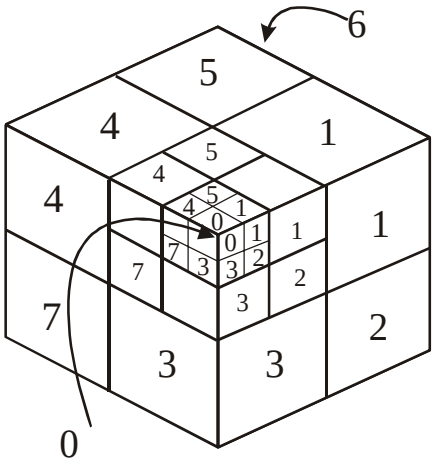


**Figure 1 [Baciu 1999]**

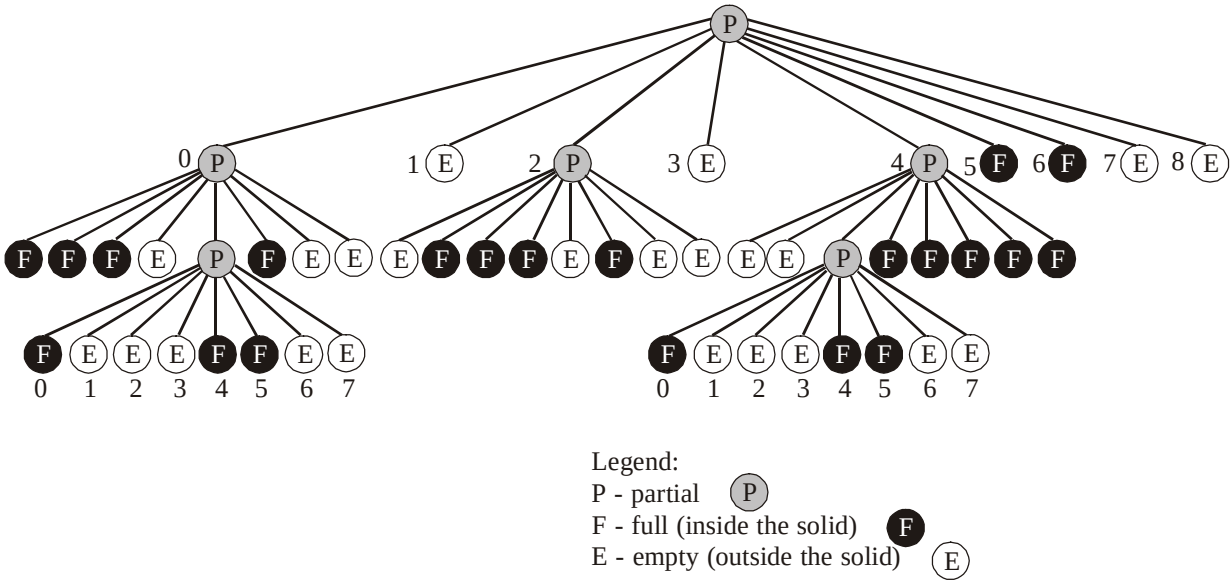


**Figure 2 [Baciu 1999]**

minimum scale are reached [Brunet 1990; Baciu 1999]. The user can fix the subdivision and the depth of the tree by specifying both the maximum and the minimum scales.

The representation of solids through octrees leads to advantages in revealing the scenes (through simplifying the algorithm of hiding the surface) and in simplifying the CSG representation (simplifying the algorithms for the boolean operations between 3D objects), when talking about 3D scenes octree representated.

## PRESENTING THE APPLICATION

The graphical application proposed for the modeling of recursion and trees concepts was made in Java using the graphical functions package gl4java, the equivalent for the OpenGL library for 3D graphics. As entring data, the application receives the b-rep representation of a solid. The representation is given as a file that contains the vertex coordinates for the triangles margining the solid. Starting from this definition of the solid, the application makes an internal representation of the solid as an octree. The solid is circumcised of a cube (Figure 3). Each node of the octree will contain a subcube, which is the outcome of the original cube's space division. The nodes belonging to these cubes, which contain only volumes that do not belong to the represented solid, will have no children. Nodes related to cubes that contain a whole volume of the solid, can be found in the same situation. Nodes that contain both parts of solid volumes and free space volumes will have eight children which will be in one of the situations presented above.

For the representation of the solid, the application will use functions for hiding parts of surface, highlighting, and so on, it already existing in gl4java library. Subcube edges are going to be represented by the application during the depth running through the tree. What can be observed is that the dividing process continues only for non-homogeneous volumes (Figure 4). The application allows visualizing the crossing in depth of the volume, rotating the image or manipulating the position of the image, bringing it closer or moving it away. The algorithm limits the level of divisions in subcubes at a maximum value. The application allows

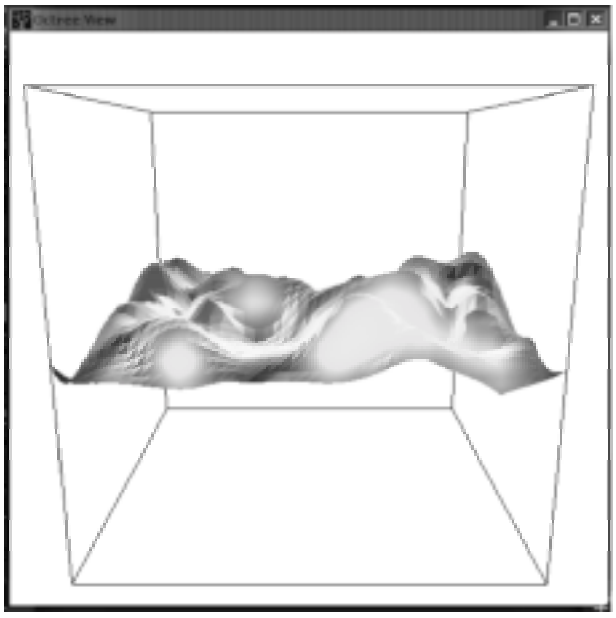

**Figure 3**

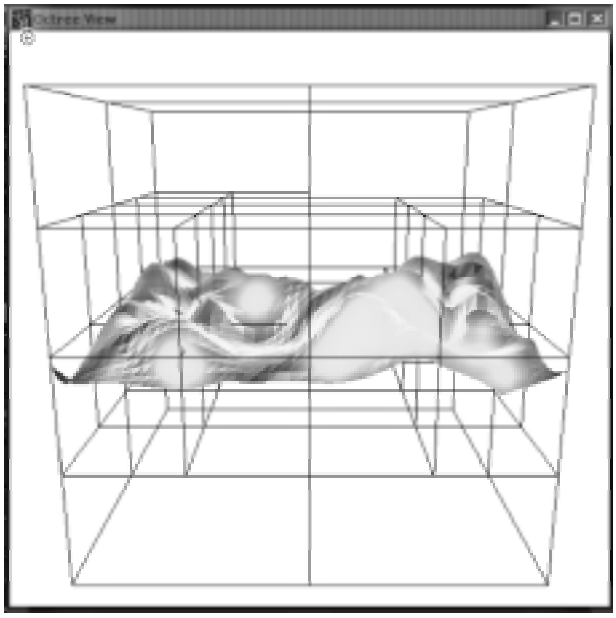

**Figure 4**

both the direct and the reverse crossing of the spatial divisions of the volume that circumcise the solid.

Modeling the concept of recursion by repeated divisions of space at the user's free will, allows a better understanding of the recursion. The exit condition from the recursive procedure is designed by dividing only parts of the cube that contain parts of the represented volume. The additional exit condition, after a limited number of levels, corresponds in graphics to the volume's division until obtaining the volume elements – voxels.

The application's interface allows navigating through the octants of the representation, visualizing them from any angle, rebuilding an octant belonging to a superior level. All these facilities are captivating to students and the understanding of a difficult abstract concept becomes an enjoyable activity.

## CONCLUSIONS

The application shown above is useful to explaining the concept of recursion within "Computers programming", subject that is taught to students in computer science in the first two years. The application is also useful to teaching concepts on solid representation within the subject "Computer Graphics Programming" which is taught in the fourth year of studies at the same department. The application hasn't yet been used in teaching but we hope that next year it will prove itself to be useful in the student's learning process.

## REFERENCES


Augenstein, M., Tenenbaum, 1976, A., A lesson in Recursion and Structered Programming, Proceedings of the ACM SIGCSE-SIGCUE technical symposium on Computer science and education, pages 17-23.

Baciu, R., Volovici, D., 1999, Sisteme de prelucrare grafică, Editura Albastră, Cluj.

Brunet, P., Nava, I., 1990, Solid Representation and Operation Using Extended Octrees, ACM Transactions on Graphics, Vol.9, No. 2, pages 170-197.

George, C., E., 2000, EROSI - Visualising Recursion and Discovering New Errors, Procedings of SIGCSE 2000 Conference, Austin, TX, USA, pages 305-309.

Götschi, T., Sanders, I., Galpin, V., 2003, Mental Models of Recursion, Proceedings of SIGCSE'03 Conference, Reno, Nevada, USA, pages 346-350.

Haberman, B., Averbuch, H., 2002, The Case of Base Cases: Why are They so Difficult to Recognize? Student Difficulties with Recursion, Proceedings of the ITiCSE'02 Conference, Aarhus, Denmark, pages 84-88.

Henderson, P., B., Romero, F., J., 1989, Teaching Recursion as a Problem-Solving Tool Using Standard ML, Proceedings of the twentieth SIGCSE technical symposium on Computer science education, Louisville, Kentucky, USA, pages 27-31.

Naps, T., L., 1989, A Java Visualiser Class: Incorporating Algorithm Visualisations into Students'Programs, Proceedings of ITiCSE '98 Conference, Dublin, Ireland, pages 181-184.

Stern, L., Naish, L., 2002, Visual Representations for Recursive Algorithms, Proceedings of SIGCSE'02 Conference, Covington, Kentucky, USA, pages 196-200.