Biodiversity Informatics, 6, 2009, pp. 5-17

AN EFFICIENT SEGMENTATION ALGORITHM FOR ENTITY INTERACTION

EUGENE CH'NG[1]

School of Computing and Information Technology, The University of Wolverhampton Wulfruna Street, WV1 1SB, Wolverhampton, United Kingdom.

Abstract.—The inventorying of biological diversity and studies in biocomplexity require the management of large electronic datasets of organisms. While species inventory has adopted structured electronic databases for some time, the computer modelling of the functional interactions between biological entities at all levels of life is still in the stage of development. One of the challenges for this type of modelling is the biotic interactions that occur between large datasets of entities represented as computer algorithms. In real-time simulation that models the biotic interactions of large population datasets, the use of computational processing time could be extensive. One way of increasing the efficiency of such simulation is to partition the landscape so that entities need only traverse its local space for entities that falls within the interaction proximity. This article presents an efficient segmentation algorithm for biotic interactions for research related to the modelling and simulation of biological systems.

Key words.— artificial life, entity interaction, individual-based model, optimisation algorithms, segmentation.

Biodiversity research studies the "variation of life at all levels of biological organisation" (Gaston and Spicer 2004) and is used to refer to "the whole range of activities traditionally connected with inventorying and studying living resources" (Lévêque and Mounolou 2003). Such studies frequently encounter large datasets of biological entities, especially in functional interactions between different levels of organisms and their effects on the modifying and shaping of the environment. As research in biodiversity becomes complex, computer modelling and simulation become a necessary means for conducting experiments so that questions that cannot be answered by traditional approaches may be resolved through the use of technology. The feasibility of computer modelling of nature is amplified in studies in biocompexity, which is defined by Lévêque and Mounolou as "the result of functional interactions between biological entities, at all levels of organization, and their biological, chemical, physical and social environments" where observational studies of complex entities above the species level such as population, biocenosis ecosystems, and biospheres require controlled experiments of their environment. For example, one may wish to project the impacts of future increase in yearly mean temperature on an ecosystem, or to attempt to computationally introduce an invasive species to see its effects on a population. In other cases, one may wish to increase the biological time and processes of a virtual ecosystem in order to predict its outcome in a hundred years' time. Other important needs are to visualise, not by charts and graphs but by utilising interactive computer graphics, the real-time interaction of a biological community in order to observe the ecosystem dynamics. In any case, there is much to be explored in the mergence of computers and biology for biodiversity research.

Traditional modelling in ecology depended upon statistical and probabilistic procedures such as classification and regression trees, generalised linear models, multivariate adaptive regression splines, and artificial neural networks. According to some comparative studies (Cairns 2001, Miller and Franklin 2002, Muñoz and Felicísimo 2004), such methods often produce inconsistent results. In

correspondence e-mail: e.chng@wlv.ac.uk

the worst case, they are fraught with errors and inaccuracies as factors considered crucial are often not being taken into account. A number of critiques have recently questioned the validity of modelling strategies for predicting the natural distribution of species. For example, a research review (Pearson and Dawson 2003) showed that many factors other than climate determine species distributions and the dynamics of distribution changes. While traditional approaches have been focused on the identification of a species' bioclimatic envelope, crucial factors such as biotic interactions (Connell 1961, Silander and Antonovics 1982, Davis et al. 1998) and evolutionary change (Woodward 1990, Davis and Shaw 2001, Thomas et al. 2001) are not taken into account and therefore making predictions erroneous and misleading. Species dispersal is another factor that has been disregarded, Pearson and Dawson (Pearson and Dawson 2003) suggest that migration limitations such as landscape barriers, deforestation, and man-made habitats can obstruct species movement and should be taken into account in such models. As incorporating these factors into the procedures of traditional approaches is difficult, it becomes necessary to explore new approaches for ecological modelling (Ch'ng 2009).

Nature has intrinsic problem-solving principles and nature-inspired design seems a potential area for discovering new methodology for research. There may not be a better way to model biological life than to simply mimic its designs by using the algorithmic format to simulate the various levels of complexities (molecular, species, ecosystem, and biosphere). This synthesis of nature formulates algorithms by extracting nature's principle rules in order that biological organisms can be replicated within the confines of voltage and silicon. Indeed, the aim of the experimental science of Artificial Life (Langton 1995) attempts to understand the mechanisms behind living systems by synthesising them so that the structure and functions of these systems may be understood and applied. One of its founders (Langton 1990) states that "By extending the horizons of empirical research in biology beyond the territory currently circumscribed by life-as-we-know-it, the study of Artificial Life gives us access to the domain of life-as-it-could-be, and it is within this vastly larger domain that we must ground general theories of biology and in which we will discover novel and practical applications of biology in our engineering endeavors."

Most, if not all of Artificial Life-based modelling techniques are Individual-Based Models (IBMs) and an extension of it called Evolutionary Individual-Based Model (EIBM) (Bornhofen and Lattaud 2006), or Agent-based models. Breckling *et al.* stated that 'IBMs contrasts with common ecological models which frequently operate on the population level and represent a population as an overall state, thereby specifying rules how the overall state changes (Breckling et al. 2005).

IBMs was identified in a visionary article by Huston *et al.* (Huston et al. 1988) as a potential model for simulating the effects of individual variation, spatial processes, cumulative stress, and natural complexities that are difficult with classical approaches: "individual-based models allow ecological modellers to investigate types of questions that have been difficult or impossible to address using the state-variable approach." Individual-Based Models have been thoroughly reviewed up to 1999 (Grimm 1999) and 2005 (Grimm and Railsback 2005) followed by a recent review of EIBM (Bornhofen and Lattaud 2006). Another review of the approach have shown that individual-based models can also benefit from concepts in Complex Adaptive Systems (Railsback 2001). These novel modelling techniques could potentially resolve issues outlined earlier, such as biotic interaction, evolutionary change, and species dispersal barriers. What distinguishes IBMs from other "individual-oriented" models that acknowledge the individual level in some way but still adhere mainly to the classical modelling paradigm? There are four criterion: (1) the degree to which the complexity of the individual's life cycle is reflected in the model; (2) whether or not the dynamics of resources used by individuals are explicitly represented; (3)whether real or integer numbers are used to represent the size of a population; and (4) the extent to which variability among individuals of the same age is considered (Uchmanski J and V. 1996, Grimm and Railsback 2005). Since the emergence phenomenon observed in IBMs reflects that of ecological systems, three properties characterises them (Breckling et al. 2005): (1) they do not exist on the level of isolated subsystems; (2) they emerge on higher levels as a result of interactions of the subsystems; and (3) new properties appear at one level of a system and

are not deducible from the observation of the lower levels units or compartments of the system. This requires the modelling of individuals rather than at the population level. Such techniques require the modelling of individual biological life, including their genotype, phenotype and dynamics. This implies the requirements for large computing resources as hundreds of thousands of calculations are performed for each individual as they react and interact with their biotic and abiotic environment. If optimisation problems associated with these models are properly developed, new opportunities for resource conservation planning, habitat and biodiversity management, mitigation strategies and studies related to the understanding of demographic problems can be initiated.

This research attempts to solve a critical problem associated with computational resources that is connected to artificial life, IBMs or agent-based modelling of biological systems at the species level and the simulation of biotic interactions among large datasets of spatially-explicit sessile organisms and a suggested extension for vagile life forms. The following sections explore optimisation techniques and formulate logical methods for optimising entity interactions. The paper concludes with a discussion of the results and opportunities for future work.

OPTIMISATION TECHNIQUES

Unlike statistical methods, which uses pixels as patch size of *n* meter of space to describe large communities of organisms on a landscape, new approaches require the modelling of individual organisms and all its internal processes and external interactions. Experience tells us that the modelling of biological life using computer algorithms requires large numbers of variables, algorithmic structures, and calculations. This is true even for a single entity. The availability of computing resources for simulation becomes an exponential challenge when biotic and abiotic interactions occur and entities reproduce. Resource bottleneck is a problem that requires solving before new modelling methods can see fruition. Unfortunately, a review of literature in the area yielded little evidence of such studies. Schulz and Reggia developed a method for predicting nearest agent distances in artificial worlds (Schulz and Reggia 2002). Other methods are developed for representing complex outdoor scenes in computer graphics (Snyder and Barr 1987, Deussen et al. 1998, Soler et al. 2003) but algorithms have not been formalised for improving the efficiency of large entity interactions.

Data structures for managing and categorising large collections of data are available. The simplest perhaps is the array data structure. More advanced structures are hierarchical data structures, which is based on recursive decomposition (Aho et al. 1974). Well known hierarchical data structures (Samet 1984) used in the science and engineering for efficient representation and improving execution times are binary trees, quadtrees, and octrees. A tree in computer science terminology emulates a tree structure. It has sets of link nodes starting from a root node and proceeds through the child nodes (branch) and finally to the leaf nodes (final nodes). A child node has one parent node and a parent node may have zero to many nodes. A binary tree contains only two child nodes and is not suitable for partitioning terrains into equal parts (where the ratio is 1). Quadtree and octrees contain only four and eight nodes respectively. As we shall soon see in the discussions, hierarchical data structures presents some problems for managing large indexes that have frequent changes. The approach presented in this article uses a data structure that is non-hierarchical. It divides terrains based on the square root of the number of segments, $\sqrt{S}$. The approach is flexible and is able to segment a landscape into an infinite number of equal parts. The next section explores some concepts related to the efficiency of segmentation logic.

EXPLORING THE LOGIC OF SEGMENTATION METHODOLOGY

The segmentation algorithm presented here is an optimisation technique developed and applied for investigating a new approach of vegetation modelling for studying palaeoenvironments (Ch'ng and Stone 2006b, a, Ch'ng 2009). The optimisation technique targets non cell-based IBMs and agent-based models. In other words, it does not apply to discrete environments such as Cellular Automata (CA), E.g., (Gutowitz 1991, Ginot et al. 2002). The IBMs must be spatially explicit and in a continuous environment. The optimisation technique targets sessile organisms (e.g., plants) but can be easily extended to include vagile life forms (e.g., fish,

animals and insects) and is suggested in the Discussions section. The following paragraphs explore segmentation logics.

In the ecosystem model, there are bound to be large numbers of entity interactions, both accessing and competing for resources. The efficiency of an optimisation technique will determine the computational speed of the simulation. In an algorithm, the entities are usually stored in a single array, in more advanced simulation requiring constant insertion and deletion (births and deaths) entities are stored within a c*ollection* data structure. In the simulation cycle, each entity accesses every other entity in the same collection to determine their proximities for interaction or competition. If entities are at proximity, interaction occurs. Therefore, if the collection contains ten simple two state entities, every entity will have to go through ten loops including itself for determining the proximity of other entities before computing the interaction. This amounts to 10^2, and if there are 1000 entities, the amount increases exponentially at $1000^2 = 1{,}000{,}000$. In large landscapes and actual models, vegetations could amount to thousands to hundreds of thousands of plants with computable variables, algorithms and calculations in each entity. This problem gives reason for developing optimisation techniques. In the initial stages of research, three theoretical concepts were compared for their efficiencies.

Memory Indices (MI)

The first concept requires that each entity remembers the indices of nearby entities (Figure 1). In this technique, each entity accesses only the indices in its memory for interaction and competition. The technique however, has serious limitations and is restricted only to sessile IBMs. We know that vegetation reproduces abundantly during the spring-summer seasons each year. In the simulation, when new plants are reproduced plant proximities in the entire collection will have to be traversed and computed to determine which plants should be in the memory of other plants. And each time a single plant dies the collection has to be traversed again to remove the index of the dead plant from the memory of nearby plants. This becomes very slow as the number of plants increases.

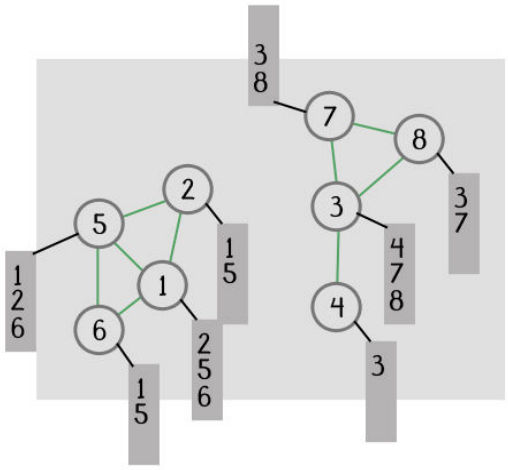

Figure 1. Inefficient Memory Indices (MI).

Collection Class Indices (CCI)

The second concept segments the landscape into different collection classes storing the indices of entities in each class (Figure 2). This is a better algorithm compared to the first since the segments in a terrain is fixed provided the landscape is not continually being re-segmented during the simulation. However, different collection classes increases memory and the need to traverse each collection during simulation wastes computational time. Furthermore, in each reproductive lifecycle, each collection has to be checked against the new plants to determine which segment boundary it belongs to. CCI differs from Cellular Automata but is very similar to the Particle In Cell (PIC) method in (Bithell and Macmillan 2007).

Segmentation Algorithm

The third technique is more efficient. In a landscape, a plant can only compete with adjacent plants at a given time and all other plants should be discounted from the interaction. In theory, the simulation time should decrease with the increase of landscape segments. We shall soon see the performance of the technique.

Figure 3a shows a landscape with groups of plants where only intersecting plants are being competed against. In the real world, computation is not a problem since interaction between entities is parallel and occurs simultaneously. However, since every virtual plant is stored within an array in a computer program, traversal is required to

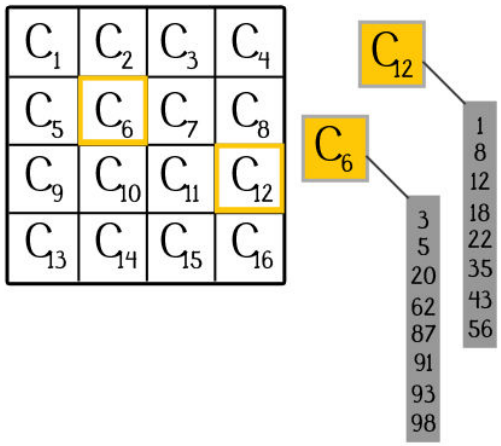

Figure 2. Inefficient Collection Class Indices (CCI).

determine which plant is 'visible' for competition. In a simulation, the ideal competition scenario for segmentation in Figure 3a can be divided into 4 segments where each plant needs only access the other plants in its own segment space (shown in *b*). In a difficult segmentation condition at *c* where the source plant (large circle) is near to a boundary or overlaps the boundary of a segment space, the source plant is required to access all other plants in neighbouring segment spaces for thorough interaction. In this case, the benefits of segmenting the landscape are not apparent as it is the same as condition *a* since plants within the entire collection is accessed. The benefits become apparent when the segments are increased (in *d*). The figure illustrates that a higher segmentation increases the speed of the program since each plant needs only access the plants within its segment space and adjacent segment spaces. The only rule needed is that the size of a segment should not be smaller than the canopy of a tree with the largest diameter.

Segmenting a landscape in a standard DirectX or OpenGL 3D coordinates is different from the 2D screen coordinate systems. The method developed in this research targets the DirectX 3D coordinate space but can be easily extended to include OpenGL and 2D coordinate spaces. Segmentation begins with divisions. The number of divisions can be from 1 to ∞ subject to the limits of computer memory. The segments are derived from the square of the divisions with $d = \sqrt{d^2} = \sqrt{S}$, where d is the number of divisions and S is the number of segments. Figure 3a contains 1 division with 1 segment, $1 = \sqrt{1^2} = \sqrt{1}$. The landscape in *b* and *c* both contain 2 divisions with 4 segments, $2 = \sqrt{2^2} = \sqrt{4}$. Figure 3d contains 3 divisions with 9 segments $3 = \sqrt{3^2} = \sqrt{9}$. The number of divisions has equal width and height. The value of the width and height of the landscape can be obtained with,

$$w_{divs} = \frac{w_{terrain}}{\sqrt{S}}$$
$$h_{divs} = \frac{h_{terrain}}{\sqrt{S}} \quad (1)$$

where w_{divs} and h_{divs} are respectively the number of divisions in the width and height of the terrain. $w_{terrain}$ and $h_{terrain}$ are the size of the width and height of the terrain. $\sqrt{S}$ is the number of divisions which divides the terrain into w_{divs} and h_{divs}.

Constructing the segments requires an understanding of the 3D coordinate system, shown in Figure 4a. The origin of the axis lies at the centre of the plane with extensions of the axis in both the negative and positive directions. The segmentation however, cannot begin from the origin but is offset in the negative (*-x,-y*) direction starting from the 'start' position show in *b*. This means that segment 0 begins from the lower left corner (*-x, -y*) and ends at the top right corner (*+x, +y*) at segment 15. The units (*-50, -25, 0, +25, +50*) are given as an example and can be replaced with any other units with equal divisions. Based on the divisions of width at *b*, it is observed that when *j=0*, *x=-50*, when *j=1*, *x=25*, when *j=2*, *x=0*, and so on. This generates a graph at *c* where an equation of the line is given (Eq. 2),

$$f(j) = j\frac{w_{terrain}}{\sqrt{S}} - \frac{w_{terrain}}{2} \quad (2)$$

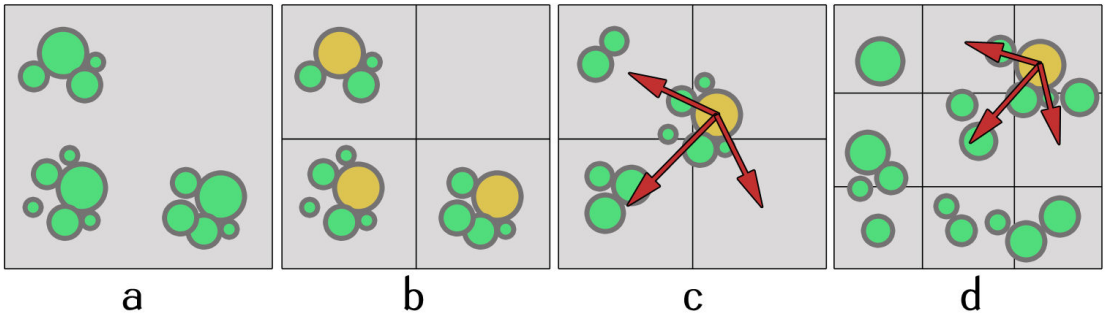

Figure 3. Efficient Segmentation Logic. A landscape with 1 segment (a), a landscape with 4 segments (b), entity accessing other segments for interaction (c), and entity interaction in a 9 segments scenario (d).

where j is the segment junction, $w_{terrain}$ is the width of the terrain, and $\sqrt{S}$ is the number of divisions. The equation offsets the first segment (0) from the origin to the 'start' position. A construction of the height divisions uses the same equation replacing $w_{terrain}$ with $h_{terrain}$.

In the algorithm, each segment is a rectangle object storing its size and position. The algorithm in standard format is given in Listing 1. *divs* is the number of divisions, *i* and *j* are each division's index, *wDiv* and *hDiv* are the width and length of each segment divided from $\frac{w_{terrain}}{\sqrt{S}}$ and $\frac{h_{terrain}}{\sqrt{S}}$.

Every plant in its own segment space accesses its adjacent segment spaces. Figure 5a shows the index accessing pattern using a one dimensional array. *T* is the target segment where a plant resides. The plant accesses its adjacent segment spaces for competition. The segmentation in *b* shows black coloured segments within the safe frame (red). In *c*, the segment within the safe frame (blue arrows) safely accesses segments that exist whereas segments outside the safe frame (red arrows) accesses non-existent segments and will generate errors in the program.

This problem can be solved by preventing the left edge and corner segments from accessing certain non-existing segments. Figure 6 shows some instances of the patterns of non-existent segments where the target segment should not have access to. The pattern illustrates that depending on where the edge segment (orange) is located (top, bottom, left, right), it should not access not less or more than three non-existent segments. The corner segments should not access not less or more than 5 non-existent segments. The algorithm for segregating the edge and corner segments is given in Listing 2. *pDivisions* is the number of divisions on the landscape, *pNumOfSegment* is the number of segments on the landscape. The rest are self-explanatory.

A UML diagram showing the class relationship between the *Segment* and *Vegetation* is given in Figure 7. Every plant has a unique *VegetationID*. Within the plant, *SegmentID* defines the segment the plant is in and has a default value of *-1* when it is in the dormant stage. It is assigned when the seed is germinated as that is the beginning of competition. The *Segment* object contains a unique *SegmentID* and an array of *PlantIndices[ ]* which stores the index of each plant within the boundary of the segment rectangle. The index of a plant can only be in the *PlantIndices[ ]* array of one segment at any given time. The pseudo code in Listing 3 shows how entities are managed during simulation time.

OPTIMISATION RESULTS

Results from the segmentation algorithm covered in the previous section were recorded on a 150m² terrain with five species of trees.

Due to the nature of the simulation (large number and fluctuations of population), it is difficult to generate a comparison chart of the increase of the segments (*1, 2, ..., n*) and for each case, the number of organisms and the simulation speed. Therefore a chart has to be shown for each individual segmentation simulation.

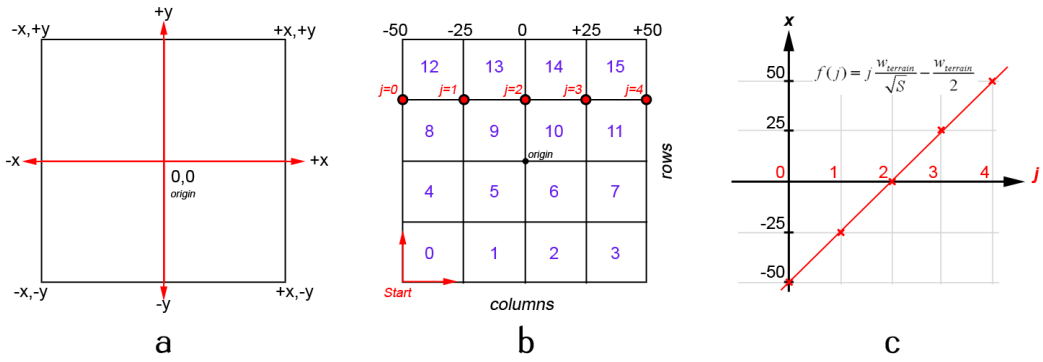

Figure 4. Segmentation constructions in 3D coordinate space (a, b, and c).

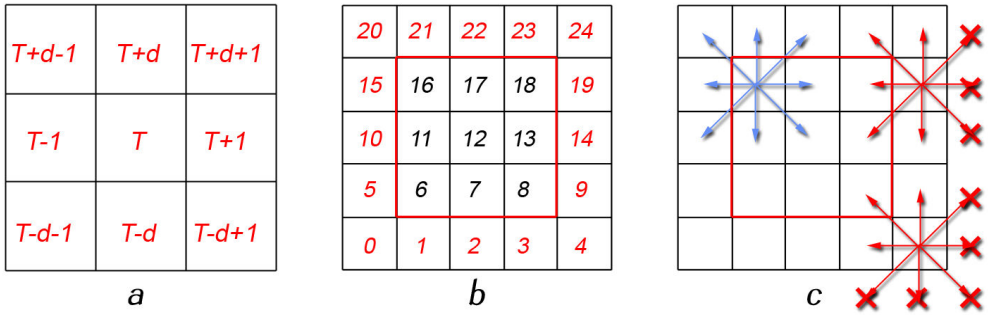

Figure 5. The concept of safe frame in segmentation algorithm: Index accessing pattern using one dimensional array (a), safe frame with black numbers (b), and frames accessing non-existent segments (c).

At the start of the simulation, the initial number of trees is 147 distributed among the species.

Three segmentation experiments were conducted to compare the results using the same settings where vegetation reproduces up to a maximum of 162 on one of the experiment. The first segmentation test uses 1 segment for the landscape with a total of 162 at peak production over 258 years, the second and third partitioned the terrain into 64 and 144 segments respectively. The 64 segments version produces 147 plants at its peak over 263 years and the 144 segments version produces up to 151 during its peak over 258 years. Even though the settings are the same for all three experiments, the number of vegetation produced is different. This is due to the interaction among vegetation and the environmental variations affecting them. This however, does not significantly affect the results as comparison is made between the speed and the number of plants produced (Figure 8-11).

A comparison of the three graphs showed that there is significant increase in speed if the segmentation algorithm is applied. In Figure 9, when the number of vegetation reaches its maximum at 140, the speed reaches only 180 milliseconds using 64 segments as compared to an average of 400 milliseconds using only 1 segment (Figure 8). If the segments are increased to 144, the speed decreased to only an average of 100 milliseconds with the same amount of vegetation (Figure 10).

Figure 11 is a comparison of efficiency of segmentation between the three experiments. The graph showed that the computational speed increases significantly between 1 and 64 segments. However, the increase in performance is not apparent between 64 and 144 segments. This is due to the small number of vegetation present in the landscape. The performance becomes obvious when the number of vegetation is greatly increased near the end of the graph. Figure 12-13 are graphs

Listing 1:

```
1. int n = 0; // segment number
2. for (int j = 0; j < divs; j++) // Rows (divided on Length)
3. {
4.   for (int i = 0; i < divs; i++) // Columns (divided on Width)
5.   {
6.     x = (terrainWidth/divs)*i - terrainWidth/2; // Set the x position (Columns)
7.     y = (terrainHeight/divs)*j - terrainHeight/2; // Set the y position (Rows)
8.
9.     // Create a new segment
10.    segment[n] = new Segment(x*2, y*2, wDiv*2, hDiv*2);
11.    n += 1; // next segment
12.  }
13.}
```

Listing 2:

```
1. // Collect Segment Corners
2. pBottomLeftCorner = 0;
3. pBottomRightCorner = pDivisions-1;
4. pTopLeftCorner = pDivisions * (pDivisions-1);
5. pTopRightCorner = pNumOfSegment-1;
6.
7. // Collect Segment Edges
8. pLeftEdge = new int[pDivisions-2];
9. pRightEdge = new int[pDivisions-2];
10.
11.for (int i=0 ; i < pDivisions-2 ; i++)
12.{
13. pLeftEdge[i] = (i+1) * pDivisions;
14. pRightEdge[i]= (pDivisions-1) + pDivisions * (i+1);
15.}
```

Listing 3:

```
Initialise Simulation:
  Generate Segments
  Test all entities
    If entity is within segment
       Assign VegetationID to segment PlantIndices based on its location
       Assign SegmentID to each entity
Simulation Cycle:
  Remove dead entities and assign new entities to segment
  Simulate entity interaction within proximity segments
```

Listing 4:

```
Initialise Simulation:
  Generate Segments
  Test all entities
    If entity is within segment
         Assign VegetationID to segment PlantIndices based on its location
         Assign SegmentID to each entity
Simulation Cycle:
  Remove dead entities and assign new entities to segment
  Assign vagile entities to segment and assign SegmentID to vagile entities
  Simulate entity interaction within proximity segments
  Simulate interaction for vagile entities (viewable entities within
segment)
```

comparing segmentation results of an environment using the same landscape. Vegetation is a mixture of trees and herbaceous plants totalling 203 in number at the start of the simulation. The herbaceous plants are used because the rate of reproduction is much faster than the trees. The 64 segments version produces 14,795 plants at its peak over 23 years and the 144 segments version

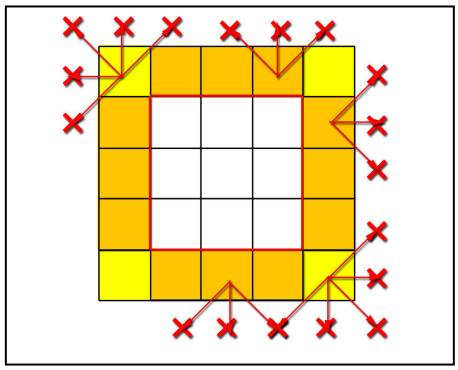

Figure 6. Accessing non-existent segments.

produces up to 12,393 during its peak over 33 years. The comparison shows a significant increase in performance using 144 segments. When the number of plants increases to around 12,000 (Figure 13), the speed decreases to only 300,000 milliseconds as compared to 64 segments at 700,000 milliseconds.

DISCUSSION

The implementation of an efficient segmentation algorithm will greatly reduce simulation time in studies dealing with large real-time datasets, such as the modelling of biocompexity where biotic interaction plays a major role in the dynamics of the system. The results in the study confirm that the use of the segmentation algorithm to divide the landscape and the devising of a strategy for managing entity interaction greatly speeds up the simulation, with an increase of >40% in all cases.

The finding suggests that increasing the segmentation decreases the speed of simulation in all studies. In the first test, (Figure 8-10), three scenarios with segments 1, 64, and 144 were compared. The number of plants in all three scenarios averaged 153. The results showed that the speed of simulation doubles from 400ms to 180ms if 64 segments were applied. If 144 segments were applied, the speed decreases from 400ms to 100 ms.

Tests conducted in a larger landscape with thousands of plants averaging 13,594 also showed a significant increase in simulation speed (Figure 12-13). When plants in the landscape reached a population of 12,000, the 144 segments yielded 300k milliseconds as compared to 700k milliseconds using the 64 segments version.

The experiments were conducted on sessile entities. But an additional cycle in the simulation will enable highly vagile entities to be included in the system. The bold lines in Listing 4 show the extension. Vagile entities may require a separate collection class from the static entities and the interaction should be after the static entities have been processed. Static objects are at priority. For example, the seeds of plants should be produced first before predators have access to them, trees should first appear before habitats are made available, and etc.

Hierarchical data structures such as quadtrees and octrees may have potential solution for such problems. But they will need to be tested. Hierarchical data structures divide a space into equal parts. Quadtrees divide a space or the subset of a space into four equal parts that are stored as nodes while octrees divide a space into eight equal parts. Apart from that, the two structures are essentially the same and achieve the same objective. These structures are found to be very efficient in a space that contains entities that do not go through large and frequent changes such as thousands of addition, deletion, interaction and movement in one simulation cycle. They are frequently used in computer graphics and visualisation. When there are large changes, it is predicted that the restructuring and traversing of the nodes will increase the simulation time. The simulation cycle in Listing 3 and 4 show a simple breadth traversal of the segments for both sessile and vagile entities. Hierarchical data structures require additional steps in each cycle to rebuild the nodes when entities die and new entities are produced. Furthermore, entities need to traverse both the breadth and depth of the nodes in order to determine their interaction proximity. In theory, such structures should not have significant increase in speed, but tests should be carried out in future work. There is much to be explored as far as efficient algorithms for entity interaction is concerned. The algorithms have produced significant results for potential interaction among

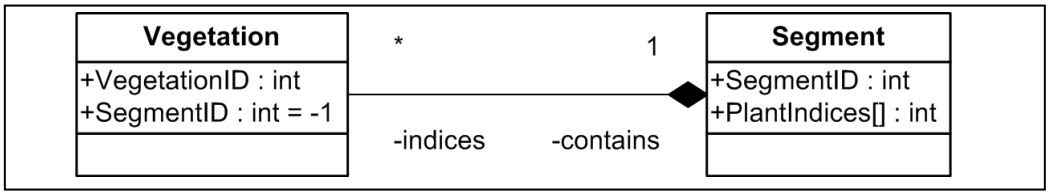

Figure 7. A UML diagram illustrating the relationship between the class Vegetation and Segment .

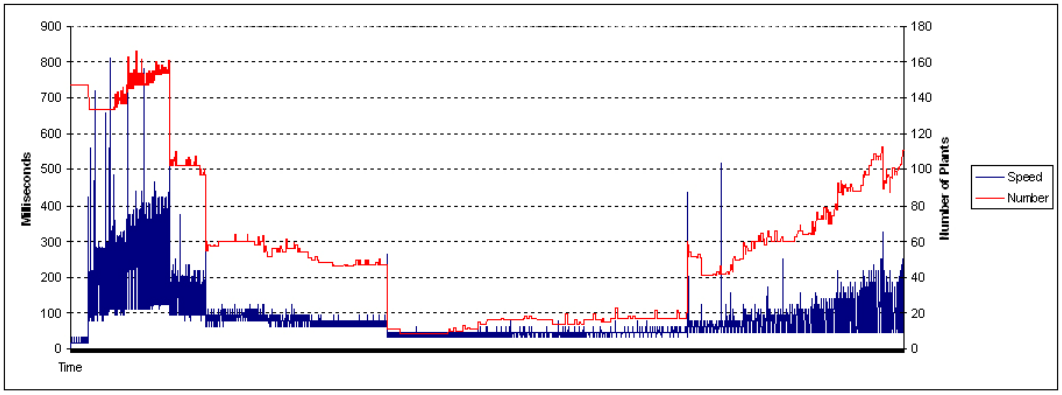

Figure 8. Experimental results using 1 segment on the landscape. The image shows the speed of simulation (the graph with high fluctuations at bottom) and the number of plants (the graph on the top).

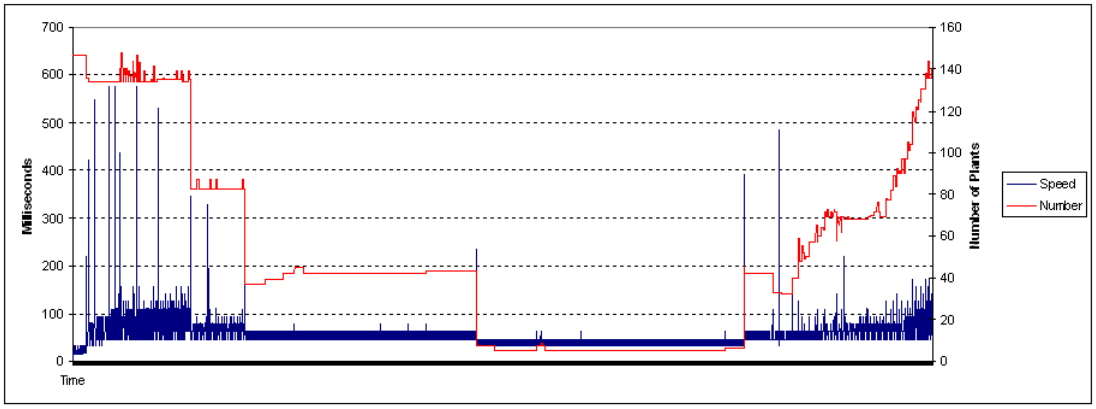

Figure 9. Experimental results using 64 segments on the landscape.

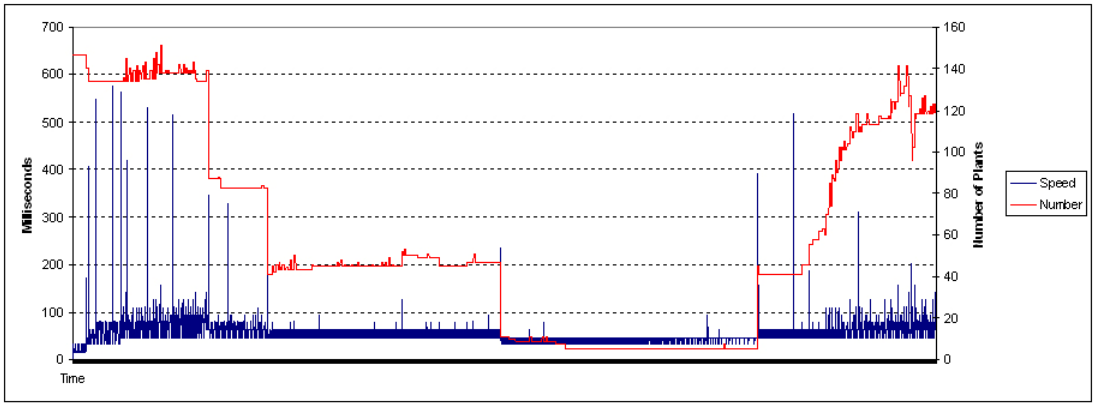

Figure 10. Experimental results using 144 segments on the landscape.

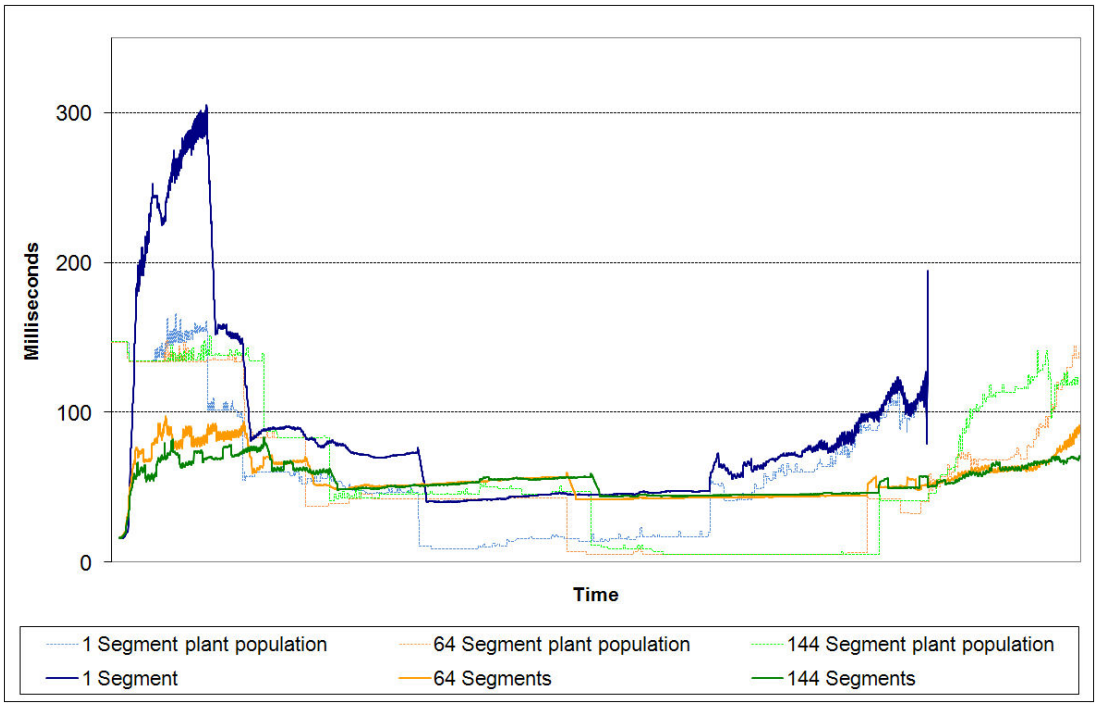

Figure 11. A comparison of the efficiency of segmentations between the experiments. Solid lines show the average speed of simulation and dashed lines of a paler shade show its corresponding plant population.

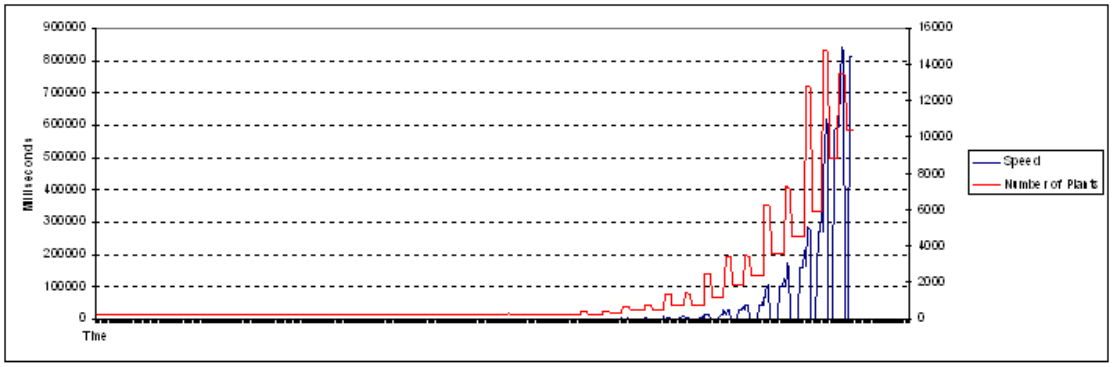

Figure 12. The comparison of efficiency of segmentation between experiments (64 segments).

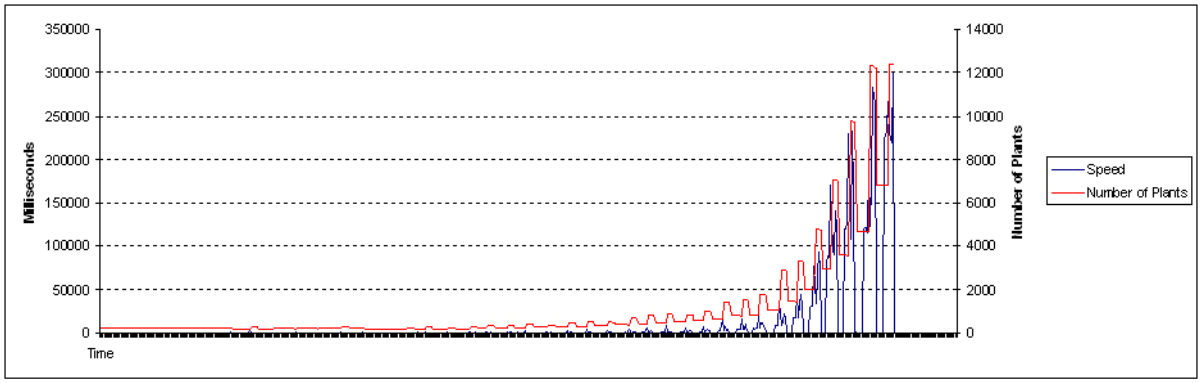

Figure 13. The comparison of efficiency of segmentation between experiments (144 segments).

thousands of entities. When vagile life forms are introduced, more efficient techniques will be required. In future work, High Performance Computing with hundreds of nodes of computer clusters is essential to support the calculations involved in biotic and abiotic interactions. Research is underway to make this a reality.

REFERENCES

Aho, A. V., J. E. Hopcroft, and J. D. Ullman. 1974. The Design and Analysis of Computer Algorithms. Addison-Wesley, Reading, Mass.

Bithell, M. and W. D. Macmillan. 2007. Escape from the Cell: Spatially explicit modelling with and without grids. Ecological Modelling **200**:59-78.

Bornhofen, S. and C. Lattaud. 2006. Outlines of Artificial Life : A Brief History of Evolutionary Individual Based Models. Lecture Notes in Computer Science, Springer-Verlag **3871**:226-237.

Breckling, B., F. Müller, H. Reuter, F. Hölker, and O. Fränzle. 2005. Emergent properties in individual-based ecological models – introducing case studies in an ecosystem research context. Ecological Modelling **186**:376-388.

Cairns, D. M. 2001. A Comparison of Methods for Predicting Vegetation. Plant Ecology **156**:3-18.

Ch'ng, E. 2009. An Artificial Life-Based Vegetation Modelling Approach for Biodiversity Research. *in* R. Chiong, editor. *to appear in* Nature-Inspired informatics for Intelligent Applications and Knowledge Discovery: Implications in Business, Science and Engineering. IGI Global, Hershey, PA.

Ch'ng, E. and R. J. Stone. 2006a. 3D Archaeological Reconstruction and Visualization: An Artificial Life Model for Determining Vegetation Dispersal Patterns in Ancient Landscapes. Pp. 112-118.*in* Computer Graphics, Imaging and Visualization (CGiV). IEEE Computer Society, Sydney, Australia.

Ch'ng, E. and R. J. Stone. 2006b. Enhancing Virtual Reality with Artificial Life: Reconstructing a Flooded European Mesolithic Landscape. Presence: Teleoperators and Virtual Environments **15**:341-352.

Connell, J. H. 1961. The influence of interspecific competition and other factors on the distribution of the barnacle Chthamalus stellatus. Ecology **42**:710-723.

Davis, A. J., L. S. Jenkinson, J. H. Lawton, B. Shorrocks, and S. Wood. 1998. Making mistakes when predicting shifts in species range in response to global warming. Nature **391**:783-786.

Davis, M. B. and R. G. Shaw. 2001. Range Shifts and Adaptive Responses to Quaternary Climate Change. Science **292**:673-679.

Deussen, O., P. Hanrahan, B. Lintermann, R. Mech, M. Pharr, and P. Prusinkiewicz. 1998. Realistic modeling and rendering of plant ecosystems.*in* Proceedings of SIGGRAPH '98 Annual Conference Series 1998. ACM Press.

Gaston, K. J. and J. I. Spicer. 2004. Biodiversity: an introduction. Blackwell Publishing (2nd Ed.), Oxford.

Ginot, V., C. Le Page, and S. Souissi. 2002. A multi-agents architecture to enhance end-user individual-based modelling. Ecological Modelling **157**:23-41.

Grimm, V. 1999. Ten years of individual-based modeling in ecology: what have we learned and what could we learn in the future? Ecological Modelling **56**:221-224.

Grimm, V. and S. F. Railsback. 2005. Individual-based Modeling and Ecology. Princeton University Press, Princeton, New Jersey.

Gutowitz, H. 1991. Cellular automata: Theory and experiment. Physica D **45**:1-3.

Huston, M., D. Deangelis, and W. Post. 1988. New computer models unify ecological theory. Bioscience **38**:682-692.

Langton, C. G. 1990. Artificial Life. Addison-Wesley Longman Publishing Co., Inc., Boston, MA.

Langton, C. G., editor. 1995. Artificial Life: An Overview. MIT Press, Cambridge.

Lévêque, C. and J. C. Mounolou. 2003. Biodiversity. John Wiley & Sons, Ltd., Chichester, West Sussex.

Miller, J. and J. Franklin. 2002. Modeling the distribution of four vegetation alliances using generalized linear models and classification trees with spatial dependence. Ecological Modelling **157**:227-247.

Muñoz, J. and Á. M. B. Felicísimo. 2004. Comparison of statistical methods commonly used in predictive modelling. Journal of Vegetation Science **15**:285-292.

Pearson, R. G. and T. P. Dawson. 2003. Predicting the impacts of climate change on the distribution of species: are bioclimate envelope models useful? Global Ecology & Biogeography **12**:361-371.

Railsback, S. F. 2001. Concepts from complex adaptive systems as a framework for individual-based modelling. Ecological Modelling **139**:47-62.

Samet, H. 1984. The Quadtree and Related Hierarchical Data Structures. Computer Surveys, ACM **16**.

Schulz, R. and J. A. Reggia. 2002. Predicting Nearest Agent Distances in Artificial Worlds. Artificial Life **8**:247 - 264.

Silander, J. A. and J. Antonovics. 1982. Analysis of interspecific interactions in a coastal plant

community - a perturbation approach. Nature **298**:557-560.

Snyder, J. M. and A. H. Barr. 1987. Ray tracing complex models containing surface tessellations. Pages 1-10 SIGGRAPH '87 Proceedings. ACM.

Soler, C., F. X. XSillion, F. Blaise, and P. Dereffye. 2003. An Efficient Instantiation Algorithm for Simulating Radiant Energy Transfer in Plant Models. ACM Transactions on Graphics **11**:204-233.

Thomas, C. D., E. J. Bodsworth, R. J. Wilson, A. D. Simmons, Z. G. Davies, M. Musche, and L. Conradt. 2001. Ecological and evolutionary processes at expanding range margins. Nature **411**:577-581.

Uchmanski J and G. V. 1996. Individual-based modelling in ecology: what makes the difference? Trends in Ecology & Evolution **11**:437-441.

Woodward, F. I. 1990. The impact of low temperatures in controlling the geographical distribution of plants. Philosophical Transactions of the Royal Society of London, Series B, Biological Sciences **326**:585-593.