Time Complexity of algorithms that update the Sierpiński-like and Modified Hilbert Curves

SANDERSON L. GONZAGA DE OLIVEIRA[1], MAURICIO KISCHINHEVSKY[2]

[1]UFLA - Universidade Federal de Lavras
DCC - Departmento de Ciência da Computação
P.O. Box 3037 - Campus Universitário 37200-000 - Lavras (MG)- Brazil
sanderson@dcc.ufla.br,

[2]UFF - Universidade Federal Fluminense
IC - Instituto de Computação
Rua Passo da Pátria, 156 - São Domingos
24210-240 - Niterói (RJ)- Brazil
kisch@ic.uff.br

Abstract. This paper presents the time complexity of two algorithms that update space-filling curves of adaptively refined domains. The Modified Hilbert (space-filling) Curve was proposed to traverse square-shaped adaptive-refined meshes. Whereas, the Sierpiński-like (space-filling) Curve was proposed in order to traverse triangular-shaped adaptive-refined meshes. Those curves are variations of the name-similar well-known space-filling curves, i.e. the Hilbert Curve and the Sierpiński Curve. Moreover, they are adapted from those classical curves that traverse regular discretized domains. This paper describes the asymptotic tight bounds of algorithms that update the Sierpiński-like and the Modified Hilbert Curves space-filling curves.

Keywords: time complexity, space-filling curves, Hilbert-like Curve, Sierpiński-like Curve.

(Received February 17, 2010 / Accepted May 21, 2010)

1 Introduction

Numerical solution of partial differential equations (PDEs) may require the use of a mesh refinement strategy that concentrates more mesh points where the solution and/or its derivatives change rapidly. For time-dependent problems, adaptive mesh refinement becomes particularly important since, by the dynamic nature of such problems, may happen migration (or occurrence) of regions that have rapid solution change.

The solution of PDEs by meshed methods, such as finite differences, volumes or elements, produces linear (or non-linear) systems. Those discrete places must be numbered so that linear systems have each row corresponding to a specific discrete place. The numbering of the discrete places is a step which can be carried out in several ways. Specifically, in [1] and [2], the authors numbered the adaptively refined square-mesh by the Modified Hilbert Curve (MHC). Whereas, in [4] and [3], the authors numbered the adaptively refined triangular-mesh by a Sierpiński-like Curve [12].

The time complexity of a calculation is measured by expressing the running time of the calculation as a function of some measure of the amount of data that is needed to describe the problem to the computer. This paper presents the time complexity of algorithms that update those space-filling curves of adaptively refined domains.

This paper deals with simple results from an ongoing work related to specific data structures that represent the adaptive mesh refinement in order to solve

PDEs by finite volume discretizations. Although simple, these results are important. Moreover, the subject has link with several disciplines, including computer graphics, computer vision, virtual reality, flight simulation, computer-aided design and geographic data processing. Specially in the numerical solution of evolutionary PDEs, the mesh must be traversed and a linear system set up and solved in each time step that the mesh is updated. Since an evolutionary PDE solution must require several time steps, traversing efficiently the mesh can help to present a complete solution much faster [3].

Related to mesh modeling, surfaces of arbitrary topology can be tessellated into a mesh of patches of discrete places. Such discrete places should be rendered very efficiently in either software or hardware. It can be used as a basic geometric primitive of the graphics pipeline [12].

One of the main disadvantages of certain meshes is that, in general, they do not provide a compact surface representation because a large number of discrete places is required to faithfully describe the geometry of a complex surface. This problem motivated the search for encoding schemes that could be used to represent adaptive refinement meshes in a more compact and efficient way [12]. The mesh encoding using strips exploits the spatial coherence of the structure. It enumerates mesh elements in a sequence of adjacent discrete places to avoid repeating the vertex coordinates of shared edges. In the traditional setting, the element strip encoding leads to the problem of converting a given mesh into the minimal set of element strips covering the mesh.

Element strips are important for accelerated rendering because they can significantly increase the throughput of the visualization pipeline. First, the data rate is increased, since only $N + 2$ vertex coordinates have to be sent to the graphics engine for a sequence of N triangles. Second, viewing operations, such as matrix transforms and clipping, need to be applied only once to the elements of the data stream, further increasing the rendering performance [12].

After this brief introduction, Section 2 describes the space-filling curves and their time complexities. Afterwards, Section 3 draws some considerations.

2 Space-filling curves

Space-filling curves or Peano Curves were first described in [8]. Intuitively, a 2D-"continuous curve" can be thought of as the "path of a continuously moving point". In [7], the author introduced a rigorous definition in order to eliminate an inherent vagueness of this notion. This definition has since been adopted as the precise description of the notion of a "continuous curve": a 2D-curve (with endpoints) is a continuous function whose domain is the unit interval $[0, 1]^2$. Moreover, the limit of the sequence given by curves of order $1, 2, \ldots$, is the curve that passes through each point in the unit square, or in a closed continuous surface. In the most general form, the range of such a function may lie in an arbitrary topological space. However, in the most common cases, the range will lie in a Euclidean space such as the 2D-plane [15]. The reader is referred to [9] for further details of space-filling curves.

2.1 The Hilbert Curve

The Hilbert Curve (see Figure 1) is known as the second proposed space-filling curve. Moreover, it is a continuous fractal space-filling curve first described in [6]. Hilbert Curves are recursively defined sequences of continuous closed plane fractal curves, which in the limit $n \to \infty$ completely fill the unit square: thus, their limit curves are examples of a space-filling curves. Its graph is a compact set homeomorphic to the closed unit interval, with Hausdorff dimension 2 [5]. Consider H_n the n-th approximation to the limiting curve. Although the Hilbert Curve grows exponentially with n since the Euclidean length of H_n is $2^n - 2^{-n}$, it is always bounded by a square with a finite area [13]. The reader is referred to [2] for further details of the Hilbert Curve.

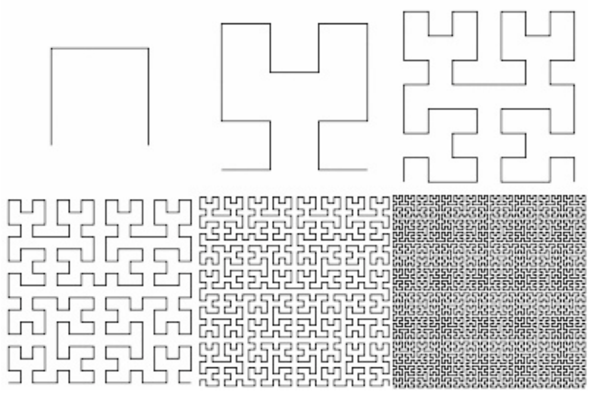

Figure 1: Six iterations of the Hilbert Curve construction

2.2 The Sierpiński Curve

The Sierpiński Curve is a space-filling curve proposed in [10] and [11]. Moreover, Sierpiński Curves are a recursively defined sequence of continuous closed plane fractal curves, which in the limit $n \to \infty$ completely fill the unit square [14]. Sierpiński Curves in uniform successive refinement in equilateral triangles are depicted in Figure 2. Fig. 3 illustrates nine levels of the Sierpiński Curve.

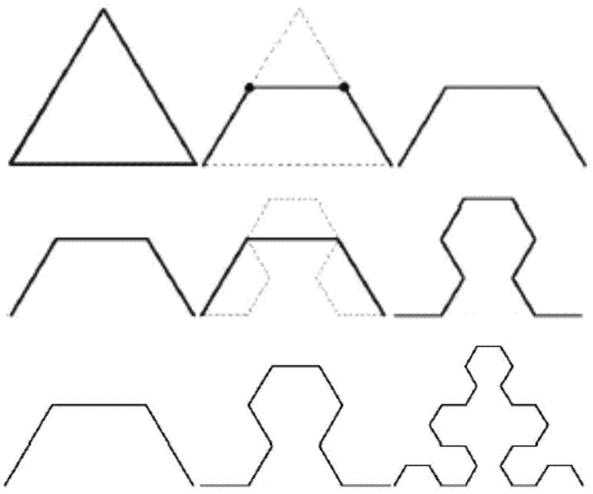

Figure 2: Generator process of the Sierpiński Curve through equilateral triangles divided into 4, 10 and 22 triangles, respectively

2.3 Space-filling curves for adaptively refined domains

In relation to space-filling curves in order to number square-shaped finite volumes in adaptively refined meshes, it seems that the first one was the Modified Hilbert Curve (MHC), proposed in [1]. Figure 4 illustrates an example of a MHC.

When it comes to space-filling curves in order to traverse adaptively refined triangular-meshes, the first one in the literature seems to be the Sierpiński-like Curve, proposed in [12] in the context of rendering, geometry compression, and theoretical investigation of paths on triangle meshes. In [4], the authors applied the Sierpiński-like Curve in order to traverse adaptively refined triangular-meshes in order to solve PDES by finite volume discretizations. Figure 5 shows an example of the Sierpiński-like Curve.

The next subsections address to time complexities of algorithms that update these space-filling curves for adaptively refined domains.

2.3.1 Time complexity of the algorithm that updates the MHC

The MHC requires to find the local shape of the Hilbert Curve for each local refinement. Namely, the local curve can have four shapes: ⊏, ⊐, ⊓ and ⊔. Algorithm 1 evaluates which local shape the new pack is, where the *<direction>SubCells* are the cells just refined and $\rightarrow$ denotes the indirection of the object.

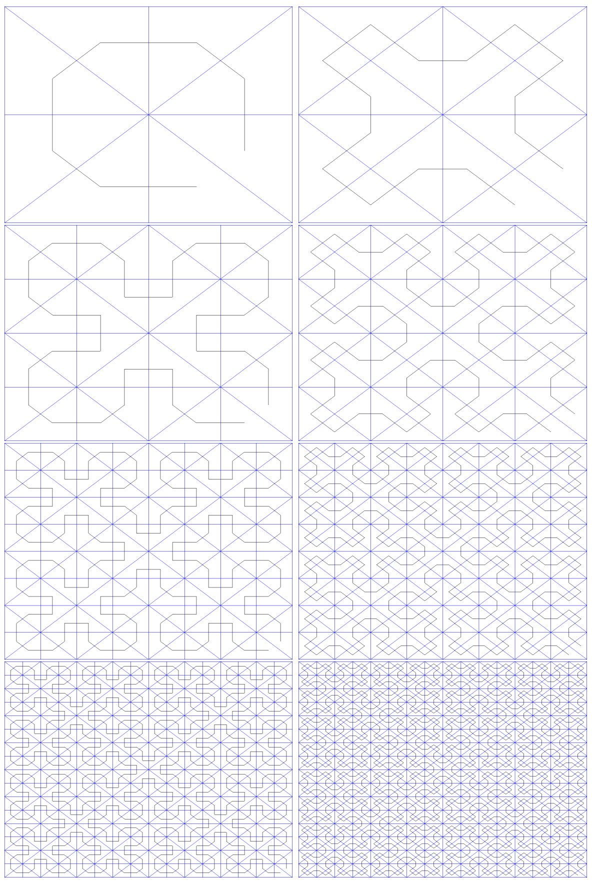

Figure 3: Sierpiński Curves with 2^3, 2^4, . . ., 2^{10} volumes

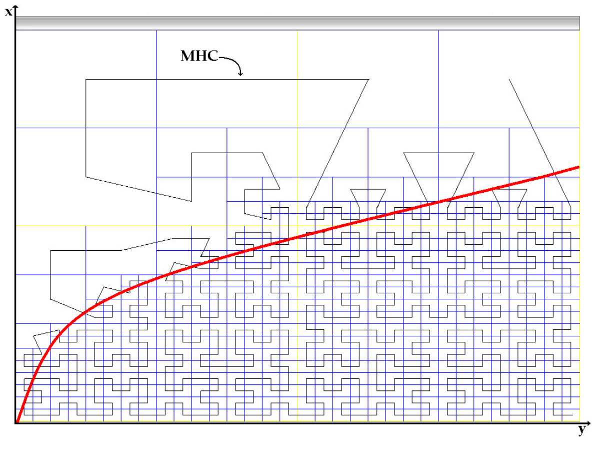

Figure 4: MHC example

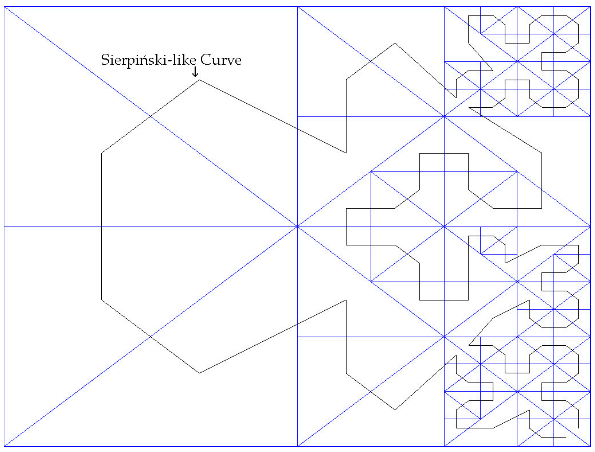

Figure 5: Example of the Sierpiński-like Curve

Algorithm 1: Update the MHC.

```
Function HilbertShape
inputs: HilbertCoordinate (integer),
        RefinementLevel (integer);
output: an integer 0, 1, 2 or 3 that represents the local
shape of the MHC;
begin
  i ← 0, j, k : integer;
  matrix hilbertTable[4][4] of integer;

  hilbertTable[0][0] ← 1; hilbertTable[0][1] ← 0;
  hilbertTable[0][2] ← 0; hilbertTable[0][3] ← 3;
  hilbertTable[1][0] ← 0; hilbertTable[1][1] ← 1;
  hilbertTable[1][2] ← 1; hilbertTable[1][3] ← 2;
  hilbertTable[2][0] ← 3; hilbertTable[2][1] ← 2;
  hilbertTable[2][2] ← 2; hilbertTable[2][3] ← 1;
  hilbertTable[3][0] ← 2; hilbertTable[3][1] ← 3;
  hilbertTable[3][2] ← 3; hilbertTable[3][3] ← 0;

  FOR k ← 1 UNTIL RefinementLevel STEP 1 DO
    /* In computing, the modulo operation finds the
    remainder of division of one number by another. */
    j ← HilbertCoordinate modulo 4;
    i ← hilbertTable[i][j];
    HilbertCoordinate ← HilbertCoordinate / 4;
  end-FOR;
  RETURN i;
end-HilbertShape;

...
numberOfHilbertShape, j : integer;
cellHilbertCoordinate, cellRefinementLevel : integer;
cellRefinementLevel ← cell→RefinementLevel;
j ← $2^{cellRefinementLevel}$;
j ← j * j;
cellHilbertCoordinate ← cell→HilbertCoordinate;
numberOfHilbertShape ← HilbertShape (
        cellHilbertCoordinate, cellRefinementLevel+1);

/* update the MHC */
IF (numberOfHilbertShape = 0) THEN /* shape: ⊏ */
  northeastSubCell→hilbertCoordinate ←
                    cellHilbertCoordinate;
  northwestSubCell→hilbertCoordinate ←
                    cellHilbertCoordinate + j;
  southwestSubCell→hilbertCoordinate ←
                    cellHilbertCoordinate + 2 * j;
  southeastSubCell→hilbertCoordinate ←
                    cellHilbertCoordinate + 3 * j;
ELSE IF (numberOfHilbertShape = 1) THEN
  /* shape: ⊔ */
  northeastSubCell→hilbertCoordinate ←
                    cellHilbertCoordinate;
  southeastSubCell→hilbertCoordinate ←
                    cellHilbertCoordinate + j;
  southwestSubCell→hilbertCoordinate ←
                    cellHilbertCoordinate + 2 * j;
  northwestSubCell→hilbertCoordinate ←
                    cellHilbertCoordinate + 3 * j;
ELSE IF (numberOfHilbertShape = 2) THEN
  /* shape: ⊐ */
  southwestSubCell→hilbertCoordinate ←
                    cellHilbertCoordinate;
  southeastSubCell→hilbertCoordinate ←
                    cellHilbertCoordinate + j;
  northeastSubCell→hilbertCoordinate ←
                    cellHilbertCoordinate + 2 * j;
  northwestSubCell→hilbertCoordinate ←
                    cellHilbertCoordinate + 3 * j;
ELSE IF (numberOfHilbertShape = 3) THEN
  /* shape: ⊓ */
  southwestSubCell→hilbertCoordinate ← cellHilbert-
Coordinate;
  northwestSubCell→hilbertCoordinate ← cellHilbert-
Coordinate + j;
  northeastSubCell→hilbertCoordinate ← cellHilbert-
Coordinate + 2 * j;
  southeastSubCell→hilbertCoordinate ← cellHilbert-
Coordinate + 3 * j;
END-IF
...
```

The HilbertShape function sets up an auxiliary matrix and performs some calculations in order to find the local shape of the MHC. It receives two integes: the Hilbert coordinate and the refinement level of the *new pack*. When a finite volume is refined, it is substituted by a new pack of finite volume. The HilbertShape function returns one integer between [0;3] that represents the local shape of the MHC. The HilbertShape function is called by a code that, depending of its return, update the MHC accordingly.

Since the first and last finite volumes of the MHC are neighbors, traversing the mesh could be seen as a Hamiltonian cicle. Notice that the Algorithm 1 does not directly *solve* the problem to traverse each finite volume once and only once, i.e. a NP-Complete problem. The Algorithm 1 just updates a curve already constructed, a concept used by space-filling curves. Moreover, one can construct, e.g. four finite volumes as an initial discretization. In the following, one can straightforwardly link these finite volumes creating the regular first level MHC, implemented as a double-linked list.

Consider that *i*, *o* and *j* are finite volumes in the sequence of the MHC. Furthermore, *i* and *o* point to each other and so *o* and *j* does. Namely, *o* is between *i* and *j* in the MHC sequence. When a single finite volume is refined, say *o*, this one is substituted by four new finite volumes, say *f*, *s*, *t* and *l*. Notice that *o* is deleted, i.e. its memory is freed, when it is refined. Algorithm 1 evaluates which of these new four finite volume must be the first one, say *f*, which must be the last one, say *l*, and the other two ones, say *s* and *t*, in the local MHC sequence. More precisely, *f*, *s*, *t* and *l* form the sequence of the new finite volumes inserted locally, i.e. a new local piece of the double-linked list MHC. The Algorithm 1 updates the links in order to *i* and *f* point to each other and so *l* and *j* does.

The Θ-notation defines an asymptotic tight bound for the growth of a function. Analysing Algorithm 1, the core of the computation is the highlighted instruction FOR in the HilbertShape function. The loop in the instruction FOR in HilbertShape function performs *RefinementLevel* times and it defines the growth order of the running time of the algorithm. Notice that the code that calls the HilbertShape function passes *cellRefinementLevel+1* as the parameter. Moreover, *cellRefinementLevel+1* indicates the level of the new pack. The time complexity analysis of this algorithm is straightforward. Clearly, Algorithm 1 $\in \Theta(L)$, where L is the level of refinements of the refined *pack*.

2.3.2 Time complexity of the algorithm that updates the Sierpiński-like Curve

Figure 6 shows a local refinement of a triangle. Since the local Sierpiński-like Curve traverses the new pack starting (or finishing) in volume d1 and finishing (or starting) in volume d2, the curve can link: A $\leftrightarrow$ d1 $\leftrightarrow \cdots$ d2 $\leftrightarrow$ B; A $\leftrightarrow$ d1 $\leftrightarrow \cdots$ d2 $\leftrightarrow$ C; or B $\leftrightarrow$ d2 $\leftrightarrow \cdots$ d1 $\leftrightarrow$ C, since it is implemented by a double-linked list.

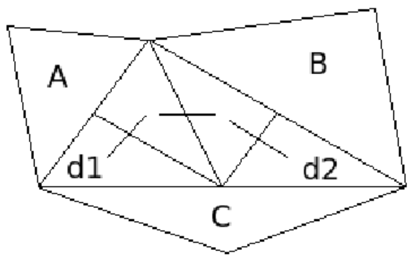

Figure 6: Volume traversing in a local triangular refinement

Algorithm 2: Update the Sierpiński-like Curve.

```
...
FaceNumber ← 3; /* constant value */
/* previous → cell2 → cell4 → cell3 → cell1 →next*/
boolean: first ← true;
integer: i, j;
/* cell2 contains the conectivity information of the local
double-linked list: each node contains pointers to the
previous and the next nodes */
/* isn't it the first volume? */
IF (cell2->previous = NULL) THEN
  /* for each edge of Cell1 */
  FOR i ← 1 UNTIL FaceNumber STEP 1 DO
    /* for each edge of the previous node */
    FOR j ← 1 UNTIL FaceNumber STEP 1 DO
      /* compares each pair of coordinates of Cell1 with
         each pair of coordinates of the previous node of
         the list */
      IF (cell1->edge[i](x,y) =
           cell2->previous->edge[j](x,y)) THEN
      /*previous→cell1→cell4→ cell3→cell2→next*/
      first ← false;
      /* jump to the next instruction after loops */
      BREAK;
    end-IF;
  end-FOR;
 end-FOR;
end-IF;
...
```

Algorithm 2 shows a piece of pseudocode that locally updates the Sierpiński-like Curve when an adaptive refinement is applied.

The piece of pseudocode in Algorithm 2 receives as input finite volumes cell1 and cell2 and finds which of them, cell1 or cell2, will be the first and the last ones in the local Sierpiński-like Curve order. Moreover, it finds which of them shares its coordinates (x,y) with the previous finite volume in the Sierpiński-like Curve already constructed. Furthermore, the proper cell2 contains the conectivity information of the finite volume that was refined. The output of this piece of pseudocode is the *first* variable, set as true or false.

The code assumes that cell2 is the first one, so it sets the variable named *first* as true. If cell1 is the first one in the local sequence, *first* is set to false and the pointers are set accordingly in the Algorithm 4.

Analysing Algorithm 2, the core of the computation is the two nested instructions FOR that define the growth order of the running time of the algorithm. Since *FaceNumber* is a constant value, the limits in the instructions FORs are constant.

Now, consider another piece of pseudocode. Algorithm 3 is another version of the Algorithm 2. Algorithm 3 verifies the consistency of the Sierpiński-like Curve in each local refinement of the mesh.

Algorithm 3 performs similarly to Algorithm 1. The piece of pseudocode in Algorithm 3 also receives as input finite volumes cell1 and cell2 and finds which of them, cell1 or cell2, is the first and the last ones in the local sequence. Moreover, the algorithm evaluates cell1 and cell2 in order to discover which of them shares its coordinates (x,y) with the previous finite volume in the Sierpiński-like Curve already constructed. Similarly to Algorithm 2, the proper cell2 contains the conectivity information of the finite volume just refined. Its pointers are updated to a finite volume of the new pack. The output of this piece of pseudocode is also the *first* variable, set as true or false.

The code also assumes that cell2 is the first one in the local sequence, so it sets the variable named *first* as true. In the following, it verifies if cell1 is the first one in the local sequence. If it is, Algorithm 3 sets *first* as false and, in the following, verifies if the next finite volume of the already constructed Sierpiński-like Curve is really cell2. Notice that the four loops are nested. In the following, the pointers are also set accordingly in the Algorithm 4.

Algorithm 3: Sierpiński-like Curve consistency.

```
...
FaceNumber ← 3; /* constant value */
/* previous → cell2 → cell4 → cell3 → cell1 →next*/
boolean: first ← true;
integer: i, j, m, n;
/* cell2 contains the conectivity information of the local
double-linked list: each node contains pointers to the
previous and the next nodes */
/* isn't it the first volume? */
IF (cell2->previous = NULL) THEN
  /* for each edge of Cell1 */
  FOR i ← 1 UNTIL FaceNumber STEP 1 DO
    /* each edge of the previous node of the linked list*/
    FOR j ← 1 UNTIL FaceNumber STEP 1 DO
      /* compares each pair of coordinates of Cell1 with
        each pair of coordinates of the previous node of
        the list */
      IF (cell1->edge[i](x,y) =
          cell2->previous->edge[j](x,y)) THEN
        /* is it the last volume? */
        IF (cell2->next = NULL) THEN
          /*previous→cell1→cell4→cell3→cell2→next*/
          first ← false;
          /* jump to the next instruction after loops */
          BREAK;
        end-IF;
        FOR m ← 1 UNTIL FaceNumber STEP 1 DO
          FOR n ← 1 UNTIL FaceNumber STEP 1 DO
            /* compares coordinates from the 2nd cell to
                the next one */
            IF (cell2->edge[m](x,y) =
                cell2->next->edge[n](x,y)) THEN
              /* previous → cell1 → cell4 → cell3 →
                  cell2 → next */
              first ← false;
              /* jump to the next instruction after loops*/
              BREAK;
            end-IF;
          end-FOR;
        end-FOR;
      end-IF;
    end-FOR;
  end-FOR;
end-IF;
...
```

Consider that each finite volume is represented by a node of a particular data structure. Algorithm 2 or 3 is followed by Algorithm 4 that links the nodes of the new finite-volume pack. The input of the piece of pseudocode described in Algorithm 4 is the *first* variable set as true or false. The output of Algorithm 4 depends on the *first* variable. It updates the pointers of the nodes to either $\cdots \leftrightarrow$ cell2 $\leftrightarrow$ cell4 $\leftrightarrow$ cell3 $\leftrightarrow$ cell1 $\leftrightarrow \cdots$ or $\cdots \leftrightarrow$ cell1 or $\leftrightarrow$ cell3 $\leftrightarrow$ cell4 $\leftrightarrow$ cell2 $\leftrightarrow \cdots$ is set.

Notice that, since cell2 already contained the conectivy information of the refined finite volume, either cell2 and the previous finite volume in the Sierpiński-like Curve point to each other if cell2 is the first one in the local sequence or cell2 and the next finite volume in the Sierpiński-like Curve point to each other if cell2 is the last one in the local sequence. Namely, one pointer of cell2 and one of its adjacent node in the already constructed Sierpiński-like Curve do not need to be updated in one and only one situation since they already point to each other.

Algorithm 4: Link the Sierpińki-like Curve.

```
...
IF (first = true) /* update the double-linked list */
  IF (cell2->next $\neq$ NULL) THEN
    cell2->next->previous $\leftarrow$ cell1;
  end-IF.
  cell1->next $\leftarrow$ cell2->next;
  cell3->next $\leftarrow$ cell1;
  cell4->next $\leftarrow$ cell3;
  cell2->next $\leftarrow$ cell4;
  cell1->previous $\leftarrow$ cell3;
  cell3->previous $\leftarrow$ cell4;
  cell4->previous $\leftarrow$ cell2;
ELSE
  cell2->previous->next $\leftarrow$ cell1;
  cell1->next $\leftarrow$ cell3;
  cell3->next $\leftarrow$ cell4;
  cell4->next $\leftarrow$ cell2;
  cell1->previous $\leftarrow$ cell2->previous;
  cell3->previous $\leftarrow$ cell1;
  cell4->previous $\leftarrow$ cell3;
  cell2->previous $\leftarrow$ cell4;
end-IF.
...
```

Likewise described, since the first and last finite volumes of the Sierpiński-like Curve are neighbors, traversing the mesh could be seen as a Hamiltonian cicle. Notice also that the Algorithm 4 and Algorithm 2 or 3 do not directly *solve* the problem to traverse, once and only once, each finite volume of the mesh, i.e. a NP-Complete problem. They just update a curve already constructed. Moreover, one can construct, e.g. eight finite volumes as an initial discretization. In the following, one can straightforwardly link these finite volumes creating the regular first level Sierpiński-like Curve, implemented as a double-linked list. When a local refinement is applied, the algorithms locally update the double-linked list accordingly.

FaceNumber denotes the number of edges of the volume. Even employing the Algorithm 3 in which there are four nested intructions FOR, in the upper bound, there are, at most, 3^4 possibilities, i.e. the limits of the loops are constant values that do not depend on any input of the algorithm. The time complexity analysis of this algorithm is also straightforward. Clearly, the algorithm that locally updates the Sierpińki-like Curve is $\Theta(1)$.

3 Consideration Remarks

This work presents the time complexity of two algorithms that perform the local update of space-filling curves that traverse adaptively refined meshes. These space-filling curves are variations of the name-similar well-known regular space-filling curves. The algorithm that implements the local update of the Sierpiński-like Curve with its $\Theta(1)$ running time beats the algorithm that implements the local update of the Modified Hilbert Curve, whose worst-case running time is $\Theta(L)$, where L is the level of refinements of a refined pack. Particularly, in [3], the author claimed that this difference is one of the reasons in which the solution of PDEs using the adaptive mesh refinement technique can be faster described with a triangular-mesh than a square-shaped mesh.

Since MHC could store in each node the level of the refinement already evaluated in order to improve its running time efficiency, a future work shall show this alteration in the algorithm that updates the MHC. Such future work shall describe experiments with a computational analysis and comparison of the running time of those algorithms as well.

Future works shall demonstrate that these space-filling curves that traverse adaptively refined meshes can be employed to advantage in solutions that generate element strip for accelerated rendering, algorithms that compute sequential discrete places for geometry compression of multiresolution models and also in theoretical investigation of paths on (hybrid) meshes. The relative gain in efficiency may be even more compelling for 3D problems.

Acknowledgements

The authors gratefully acknowledge CAPES for the financial support and Prof. Denise Burgarelli for sent us the program code to solve the Laplace Equation by quadrangular-shaped finite volume discretizations.

References

[1] Burgarelli, D. *Modelagem Computacional e Simulação Numérica Adaptativa de Equações Diferenciais Parciais Evolutivas Aplicadas a um Problema Termoacústico*. PhD thesis, Pontifícia Universidade Católica do Rio de Janeiro, August 1998.

[2] Burgarelli, D., Kischinhevsky, M., and Biezuner, R. A new adaptive mesh refinement strategy for numerically solving evolutionary PDE's. *J. of Computational and Applied Mathematics*, 196:115–131, 2006.

[3] Gonzaga, S. L. O. *Graph-based adaptive simplicial-mesh refinement for finite volume discretizations*. PhD thesis, Universidade Federal Fluminense, January 2009.

[4] Gonzaga, S. L. O. and Kischinhevsky, M. Sierpinski curve for total ordering of a graph-based adaptive simplicial-mesh refinement for finite volume discretizations. In *XXXI CNMAC*, pages 581–585, September 2008.

[5] Hausdorff, F. Dimension und äusseres mass. *Mathematische Annalen*, 79(1-2):157–179, March 1919.

[6] Hilbert, D. Über die stetige abbildung einer linie auf ein flächenstück. *Mathematische Annalen*, 38:459–460, 1891.

[7] Jordan, C. Courbes continues. *Cours d'analyse*, pages 587–594, 1887.

[8] Peano, G. Sur une courbe, qui remplit toute une aire plane. *Mathematische Annalen*, 36(1):157–160, 1890.

[9] Sagan, H. *Space-Filling Curves*. Springer-Verlag, 1994.

[10] Sierpiński, W. Sur une nouvelle courbe continue qui remplit toute une aire plane. *Bulletin de l'Académie des Sciences de Cracovie A*, pages 462–478, 1912.

[11] Sierpiński, W. Sur une courbe cantorienne qui contient une image biunivoque et continue de toute courbe donee. *Comptes Rendus de l'Académie des Sciences*, 162:629–632, 1916.

[12] Velho, L., Figueiredo, L. H., and Gomes, J. Hierarquical generalized triangle strips. *The Visual Computer*, 15(1):21–35, 1999.

[13] Wikipedia. Hilbert curve. Technical report, Wikipedia, January 2010. http://en.wikipedia.org/wiki/Hilbert_curve.

[14] Wikipedia. Sierpiński curve. Technical report, Wikipedia, January 2010. http://en.wikipedia.org/wiki/Sierpiński_curve.

[15] Wikipedia. Space-filling curve. Technical report, Wikipedia, May 2010. http://en.wikipedia.org/wiki/Space_filling_curve.