symmetry

MDPI

Article

Hierarchical Hexagonal Clustering and Indexing

Vojtěch Uher [1,*], Petr Gajdoš [1], Václav Snášel [1], Yu-Chi Lai [2] and Michal Radecký [1]

[1] Department of Computer Science, VŠB-Technical University of Ostrava, Ostrava-Poruba 708 00, Czech Republic; petr.gajdos@vsb.cz (P.G.); vaclav.snasel@vsb.cz (V.S.); michal.radecky@vsb.cz (M.R.)
[2] Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology, 43, Sec.4, Keelung Rd., Taipei 106, Taiwan; yu-chi@mail.ntust.edu.tw
* Correspondence: vojtech.uher@vsb.cz

Received: 25 April 2019; Accepted: 23 May 2019; Published: 28 May 2019

Abstract: Space-filling curves (SFCs) represent an efficient and straightforward method for sparse-space indexing to transform an *n*-dimensional space into a one-dimensional representation. This is often applied for multidimensional point indexing which brings a better perspective for data analysis, visualization and queries. SFCs are involved in many areas such as big data analysis and visualization, image decomposition, computer graphics and geographic information systems (GISs). The indexing methods subdivide the space into logic clusters of close points and they differ in various parameters including the cluster order, the distance metrics, and the pattern shape. Beside the simple and highly preferred triangular and square uniform grids, the hexagonal uniform grids have gained high interest especially in areas such as GISs, image processing and data visualization for the uniform distance between cells and high effectiveness of circle coverage. While the linearization of hexagons is an obvious approach for memory representation, it seems there is no hexagonal SFC indexing method generally used in practice. The main limitation of hexagons lies in lacking infinite decomposition into sub-hexagons and similarity of tiles on different levels of hierarchy. Our research aims at defining a fast and robust hexagonal SFC method. The Gosper fractal is utilized to preserve the benefits of hexagonal grids and to efficiently and hierarchically linearize points in a hexagonal grid while solving the non-convex shape and recursive transformation issues of the fractal. A comparison to other SFCs and grids is conducted to verify the robustness and effectiveness of our hexagonal method.

Keywords: space-filling curve; point clustering; Gosper curve; hexagonal grid

1. Introduction

A space-filling curve (SFC) represents a bijective transformation of multidimensional points onto 1D representation by computing an index (hash code) for each point defining the linear order of points in the space [1]. A SFC hashing algorithm usually follows a pattern of some tree or pyramid structure. SFCs are utilized, e.g., for space indexing and range querying [1–3] and also for building spatial hierarchies especially for parallel platforms [4,5]. While the SFCs based on orthogonal and triangular uniform grids are well investigated and preferred in the applications, we propose an effective and robust hexagonal Node-Gosper SFC based on the Gosper fractal [6] (Figure 1). This paper analyzes the problems of hierarchical hexagonal indexing and proposes solutions of issues related to transformations of multi-resolution hexagonal grids and fractal shape of Gosper tiles. Our novel method overcomes the lacks of naive Gosper-based approaches [7,8] and competes well with other non-hexagonal SFCs.

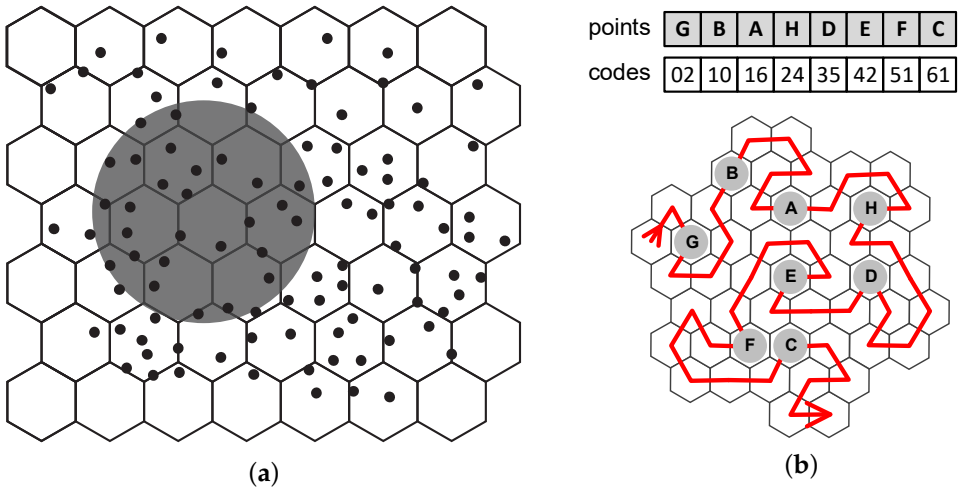


Figure 1. Hexagonal clustering: (**a**) Circular disc query. (**b**) Points ordered along the Node-Gosper space-filling curve according to the corresponding cluster codes.

A point cloud (PC) is a collection of points of the same type (2-3D real points) [9] representing a discrete form of real objects. Beside traditional PCs produced mostly by 3D scanners [9], a set of points can also represent general spatial information [10] such as location, GPS coordinates on the map [11], swarm particles [12], nodes of graph generated by visualization algorithms [13], etc. We focus on 2D point clouds as they are highly investigated in the areas such as data mining [2,3,10], computer graphics [14,15], image analysis [16–18], geographic information systems (GISs) [19–23], sensor networks [24,25], triangulation [26–29] and data visualization [12]. In the mentioned application fields, the point indexing/clustering algorithms are crucial, e.g., for efficient organization of data in the memory, analyzing their properties, visualization of clusters, computation of forces in particle systems and searching near neighbors. Point indexing algorithms decompose the space into subspaces or tiles [1,10] containing a subset and create structures for effective addressing of the points. Usually, the structures utilize the uniform regular grids that are represented by list of clusters [30], or the clusters are organized in the hierarchical manner to reduce the costs of point queries [2,10] by omitting irrelevant clusters such as empty or distant ones (Figure 1a). The hierarchies offer good query times, esp. for sparse datasets, but they are often memory consuming and unbalanced [5,31]. Linear methods save memory usage and provide constant query time for uniformly distributed datasets [30]. However, they are inefficient for sparse point clouds.

The space-filling curves combine both approaches as they define the linear order of clusters and preserve the hierarchical information about points in the hash codes. To compute an index uniquely addressing a point in a hierarchy covering the whole space, the multi-resolution grids are often used. Most widely used SFCs including Peano, Z-order, Hilbert and Sierpiński SFC [1,32–34] are for orthogonal or triangular grids. The multi-resolution hexagonal grids were utilized for image processing [18,35–37]. Burt [38] proposed a structure for coding binary images with hexagonal grids and Gibson and Lucas [39] extended this structure to the more sophisticated Generalized Balanced Ternary (GBT). Those methods serve to decompose and represent images with pixels having positive coordinates, but they do not offer any general method for real points indexing and querying. Several papers discussed the problems of different grid systems and their mutual conversion to combine their best properties [21–23], but they all miss the correct hierarchical 1-to-7 refinement in the hexagonal grid (Figure 2) for a correct hexagonal SFC. During the last years, the hexagons have gained increased interest in scientific areas including geographic information systems [19–23,40,41], data clustering and querying [36,38,39,42,43], data visualization [44,45], computer graphics [46] and computer or sensor networks [24,25]. Recently, the transportation network company Uber Technologies Inc. introduced a discrete global grid system [11] based on multi-resolution hexagonal grids. They use it to localize

cars and customers on the map, to efficiently optimize ride pricing and dispatch and to visualize and explore spatial data. The GPS coordinates of locations on the map represent the points that are indexed by hexagonal grids. The hexagons also form the basis for analysis of the Uber marketplace, they can easily approximate zones defined by edges of required area on the map and they minimize the quantization error [11]. Unlike squares and triangles, hexagons have uniform distance between neighboring cells and they have the optimal perimeter/area ratio which leads to good approximation of circles [47].

While the hexagonal grids have a wide range of applications and a convenient linear representation could simplify performing point indexing, the hexagonal SFCs are not utilized very often. The main obstacle is that a hexagon is not a rep-tile. A rep-tile is a tile which is infinitely decomposable into similar tiles. A hexagon can be decomposed only into triangles, thus the hierarchy has to be defined by bottom-up composition of hexagons creating a system of multi-resolution grids [10]. After exploring different hexagonal patterns, we decided to follow the concept of the Gosper fractal (or Flowsnake) which is a self-similar recursive fractal discovered by William Gosper in 1973 and introduced by Martin Gardner in 1976 [6,48] (Figure 2). The fractal connects the vertices of hexagonal lattice and creates a pattern of 7 hexagons called the Gosper island. These islands can be composed again into 7-pattern to create an island of greater level (Figure 2). We define a procedure extending the Gosper fractal to its node form (Node-Gosper curve) for point indexing in the manner of SFCs. The key tasks are: point localization in the hierarchy and inverse point mapping algorithm. The first published method [7] simply decomposes the points into 7 disjoint areas in the top-down manner to approximate the hexagonal 7-pattern. The paper [7] shows that the top-down decomposition is not reliable and it cannot produce a correct hexagonal hierarchy. The Gosper islands are non-convex fractal tiles (Figure 2) with jagged edges that are fitted into surrounding ones. This leads to point mislocalization in the hierarchy (esp. at the edges) which produces many non-hexagonal clusters in the lower levels of decomposition. Thus, the hexagonal properties are not preserved. Later, the inverse bottom-up mapping procedure fixing those problems was theoretically proposed [8], but its mapping algorithm is inaccurate and inefficient. To the best of our knowledge, there is no straightforward hashing algorithm mapping the points onto the Gosper curve.

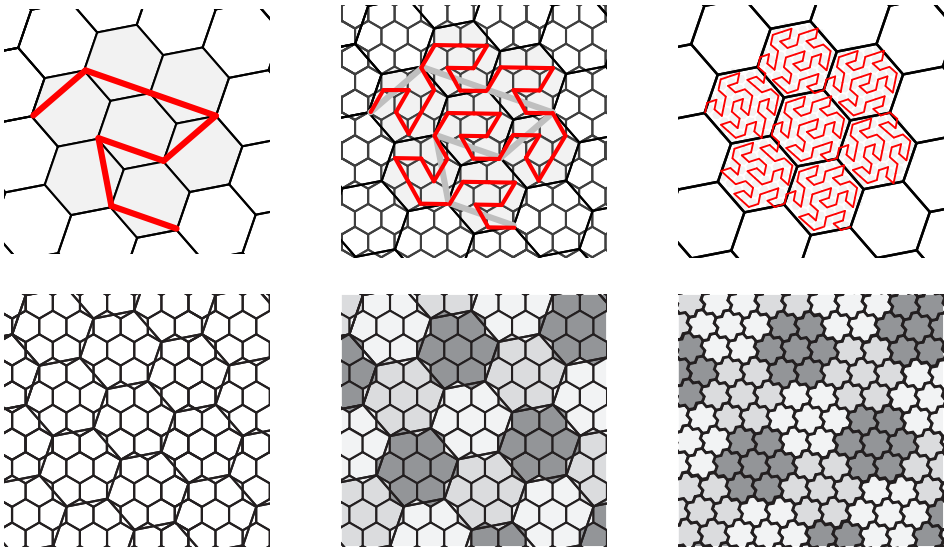


Figure 2. Row 1: Three iterations of the Gosper curve. Row 2: Tiling with Gosper islands. Seven joined hexagons represent the basic level-1 island.

This paper summarizes a background about hexagonal grids and Gosper fractal to understand the advantages and lacks of hexagons and hexagonal multi-resolution grids. The main contribution of this work lies in the derivation of the hexagonal space-filling curve solving the lacks by using the pattern of

the Gosper fractal to make a hexagonal linearization available to wide range of applications. We focus on correct point localization in the Gosper islands, stable point mapping onto the Gosper curve, fast determination of related hexagonal pattern, its rotation and indexation and efficient design of these algorithms. Our Gosper SFC method is supported by several theoretical metrics, e.g., Honeycomb conjecture [47] proving good coverage of a circle by hexagonal grid, distance metrics [49] declaring good real distance between points neighboring along the curve and construction times comparison. The metrics are verified by practical experiments comparing different grids and SFCs which show that our Node-Gosper SFC is robust and it produces the correct hexagonal hierarchy without mislocalization much faster than previous approaches [7,8].

The paper is organized as follows. Section 2 describes the related theoretical background we need for hierarchical hexagonal clustering. Section 3 gives the details of our hierarchical hexagonal composition and the corresponding Node-Gosper SFC construction. Finally, Section 4 evaluates our SFC by comparing against other grids and space-filling curves.

2. Background of Hierarchical Hexagonal Clustering

This section summarizes the theoretical and mathematical background needed to define and understand our Node-Gosper SFC. At first, we state why we prefer hexagonal grids over orthogonal and triangular ones (Section 2.1) and briefly survey selected properties of hexagonal grids. Section 2.2 overviews the existing SFCs and defines the criteria that we generally expect from a good SFC. Afterwards, we include a brief analysis of hexagonal multi-resolution grids and explanation why the Gosper fractal is selected for point indexing (Section 2.3).

2.1. Hexagonal Grids

In mathematics, a tiling means decomposition of the space into collection of geometric shapes representing some logical subspaces covering the whole space. There are only three types of regular tilings [10]: Triangular, square and hexagonal (Figure 3). Unlike other tilings, hexagons have uniform distances between tiles, a hexagon is the regular shape closest to the circle and one lattice vertex is shared by only three hexagons. The outstanding circular coverage by hexagonal grids is supported by several theoretical metrics including isoperimetric inequality [50], Honeycomb conjecture [47] and circle packing problem solution [51] which show that the best regular shape having the maximal area with the least perimeter is the hexagon. The chart in Figure 4 confirms experimentally the predominance of the hexagonal grid over the orthogonal grid. These properties are useful for disc queries and nearest neighbors search as the number of tested tiles should be minimized [42]. On the other hand, hexagons are not infinitely decomposable. We solve this by following the principles of Gosper fractal that gives hexagons a hierarchical organization.

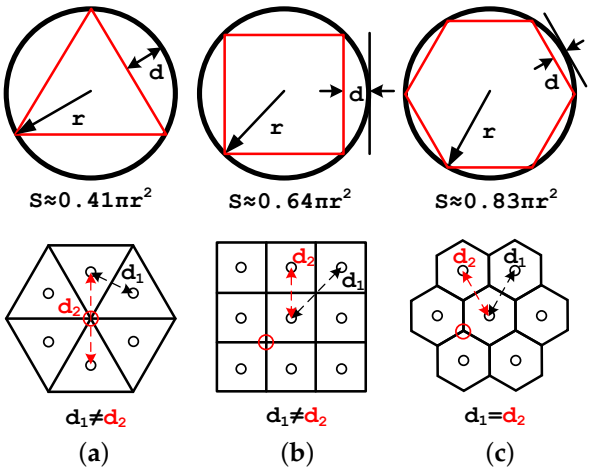


Figure 3. A comparison of properties of regular grids: (**a**) Triangular, (**b**) orthogonal, (**c**) hexagonal.

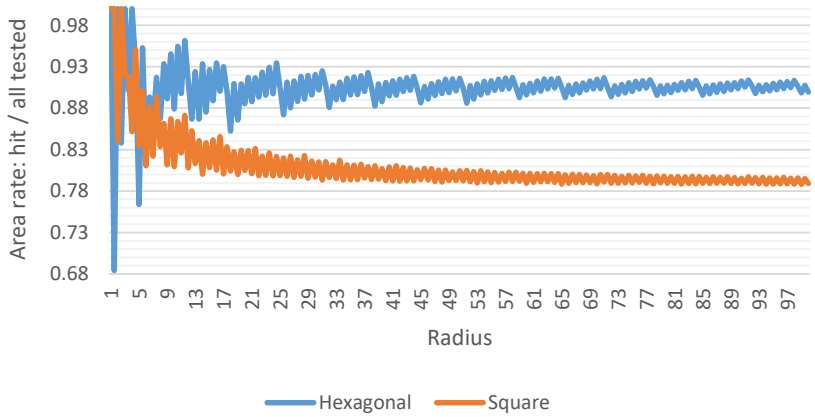


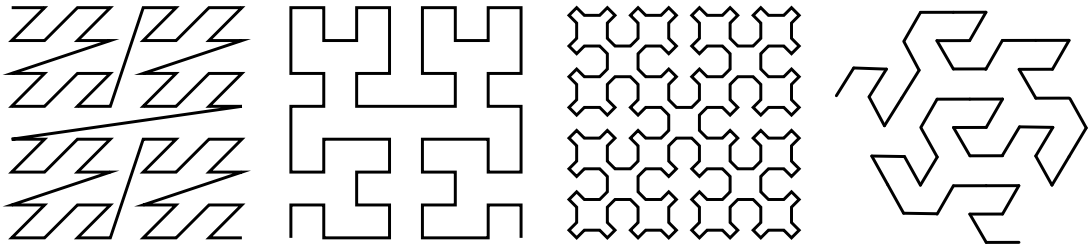
Figure 4. For a disc of radius from 1 to 100, the graph represents the rate between the area of hit clusters and all tested clusters. The area of each cluster (hexagon or square) is unitary. The tested clusters represent the corresponding complete cluster rings covering the query disc.

As the points in a PC are real points, the discrete hexagonal coordinate system has to be defined to localize them in the hexagonal grid and give them integer coordinates of the corresponding tile. The axial $(q, l) \in \mathbb{Z}^2$ and cube $(x, y, z) \in \mathbb{Z}^3$ coordinates are used to address the hexagons. The cube coordinates extend the axial ones by one redundant coordinate to get the six-directional system simplifying some calculations. If the condition $x + y + z = 0$ is met, the axial coordinates are calculated simply by neglecting the redundant y coordinate $q = x, l = z$. We refer to papers [43,52] for more details and visualizations.

2.2. Space-Filling Curves

Space-filling curves represent a family of point hashing algorithms which define the basic standards of space linearization [1]. These standards are carefully taken into account when designing our Node-Gosper SFC (see, e.g., Figure 1b). Space-filling curves transform a general n-dimensional point into a one-dimensional (1D) index by localizing it in multi-resolution grids (Figure 5). For each point, a set of hierarchical indices addressing the sub-clusters containing the hashed point is computed. The hash code is usually a bit string merged from the indices in the top-down manner so that the root index occupies the most significant bits [1]. After sorting the points by codes an SFC is constructed and the points from the same cluster (with the same hash) are aligned in the memory. An SFC order of clusters represents the depth first passage of the corresponding tree (Figure 5). The nearby points are stored close to each other in the memory. However, there can be some points lying very close on the curve, but in reality, they are far away (see, e.g., Figure 5a). The shape of clusters and their indexation are varied, and thus, there are many different curves. The popular ones are Z-order, Hilbert or Sierpinski space-filling curves [1,32–34]. SFCs are very straightforward and efficient methods for sparse-space clustering. Such a structure design is also much more convenient for massive parallel processing of clusters [4,5,12,13,53].

The linear point memory representation is used, e.g., for space indexing and range querying [1–3] or sophisticated triangulation algorithms [26,27]. SFCs are often utilized for building spatial hierarchies especially for parallel platforms [4,5]. The linearly stored points can be grouped together in parallel to build a space tree (e.q. Octree) instead of a sequential particle insertion [53]. Papers [12,13] discuss the utilization of SFCs for speeding up the nearest neighbors (NNs) algorithm and real time graph layout visualization. To bring a solid hexagonal method, we define the general criteria for good SFCs [1] in Section 2.2.



(**a**) (**b**) (**c**) (**d**)

Figure 5. The examples of space-filling curves: (**a**) Z-order, (**b**) Hilbert, (**c**) Sierpiński, (**d**) Gosper.

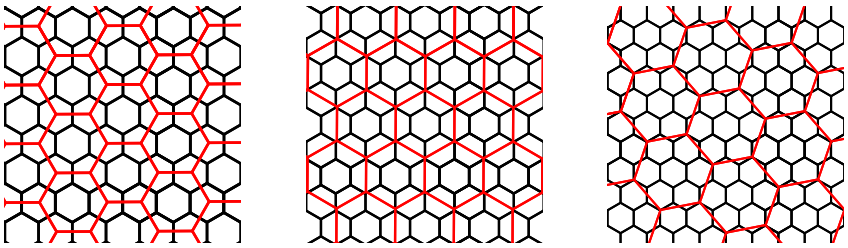
Space-Filling Curves Criteria

The SFCs differ in their properties, so that there is no SFC that works efficiently for all kinds of applications. However, several criteria should be met to define a high-quality SFC [1]. The conditions 1–3 are mandatory followed by quality criteria:

1. Mapping between a point and SFC code is bijective—each point is assigned a unique hash code defining the cluster along the SFC. This has to be possible for greater depths of hierarchy.
2. The curve is space-filling—every point of space lies on the curve.
3. Total ordering—each cluster is visited by an SFC only once.
4. Mapping between points and codes should be easy to compute.
5. Two points close along the curve should be also close in the multidimensional space.
6. Sequentialisation of a contiguous data set should also lead to a contiguous sequence of code numbers. This is handy for efficient memory addressing.

2.3. Hexagonal Curve Selection

To best fulfill the requirements for good SFCs, many hexagonal and combined fractal tiling patterns have been studied (see, e.g., [17,37,38,42]). The fractal patterns usually have very complex shapes, and thus, they are not appropriate for point clustering. There are three reasonable variants of multi-resolution hexagonal grids often used in the Discrete Global Grid Systems [21–23,41,54]. Generally, the grids of aperture 3 and 4 are preferred (see Figure 6) to avoid the problems with fractal shapes. While these grids can be used for localization, they do not preserve the basic properties of the regular hexagonal grids. A grid of level $i \in \mathbb{N}\backslash\{0\}$ divides the hexagons of level $i+1$ into non-hexagonal polygons, which have to be joined together on the level $i+1$. These hierarchies are incongruent, and therefore, the classical tree-based algorithms cannot be utilized with them. Thus, only the hexagonal grid of aperture 7 (Figure 6C) is applicable for an SFC construction (Figure 2) because the composites are completely disjoint (not overlapping). As a result, the points of each island can be aligned and linearly stored in the memory in the manner of an SFC.



(**a**) (**b**) (**c**)

Figure 6. Multi-resolution hexagonal grids: (**a**) aperture 3, (**b**) aperture 4, (**c**) aperture 7.

As mentioned before, the Gosper curve is a curve based on such a hierarchy. The space filled by the i-th recursion of this curve is bounded by the level-i Gosper island (Figure 2). The Gosper curve and its islands have many notable properties deriving from the essence of the hexagonal grid and its distance characteristics (see Figure 3) [10,37,42]. The Gosper curve satisfies the conditions of the good SFCs such as the continuity, uniformity, scale invariance, etc. [49,55] (see Section 2.2). Each point in the space is visited just once, and the distance between adjacent clusters along the curve is uniform. Any hexagon (or Gosper island) shares edges with six other identical shapes so that the Gosper islands can be simply addressed by the same coordinate system as the regular hexagons. All the points of the dataset lie on the curve. The whole point cloud just has to be fitted into the circle inscribed into the Gosper island. The mapping is bijective for sufficiently deep hierarchies. The specific indexation of clusters and mapping complexity depend on the final algorithm. The Gosper curve is also held up by several theoretical metrics. The worst-case locality value (WL) [49] (smaller is better) examines the rate between the Euclidean distance of two points in the real 2D space and the distance along an SFC. The Gosper curve ($WL = 6.35$) is comparable with sophisticated non-hexagonal SFCs, e.g., Hilbert ($WL = 6$) or Sierpiński ($WL = 4$) curve. The *Arrwwid* number [42] (smaller is better) represents the maximum tiles hit by reasonably small radius and judges the suitability of a tiling (or an SFC) for a circular neighborhood querying. The Gosper curve has optimal $Arrwwid = 3$ while other named SFCs have at least $Arrwwid = 4$.

3. Node-Gosper Space-Filling Curve

We mentioned the main issues related to hexagonal hierarchy such as impossibility of recursive decomposition and fractal shape of Gosper islands that make it difficult to design a correct point mapping algorithm. The solutions proposed by previous work are insufficient. The approximative top-down decomposition [7] following the fractal drawing algorithm produces deformed polygons and obviously incorrect point localization. The naive bottom-up approach [8] solving the issue with point localization and producing the correct Node-Gosper SFC proposes only a theoretical pipeline with bad accuracy and performance.

This section represents the main contribution of the paper addressing all the named issues related to hierarchical hexagonal point indexing. We improve the existing simple concepts from papers [7,8] and define the complete hexagon-based system in detail with mathematical definitions and pseudocodes.

At first, the mathematical definition and the recursive construction of the Gosper fractal is reminded in Section 3.1. Following the transformations defined for the Gosper fractal, we introduce our Node-Gosper indexation model in Section 3.2 which gives the right order to the hashed points. Next, we define the procedure computing a hexagon centroid according to its hash code based on the indexation model and top-down drawing algorithm of the Gosper fractal. Section 3.4 defines our novel mapping algorithm representing the hashing function which correctly maps a point onto the Node-Gosper curve and returns the corresponding hash code.

3.1. Gosper Fractal

Our Node-Gosper SFC is inspired by the Gosper fractal drawing algorithm which is reminded here. The Gosper fractal is based on the pattern of seven hexagons (Figure 7a). The fractal is recursively constructed by replacing the oriented red arrows with the properly rotated and scaled pattern in the top-down manner (Figure 7b). The arrows connect the vertices of the hexagonal grid, and thus, this iterative method secures the geometrical continuity of the patterns and draws the correct image of the fractal. Alternatively, the same procedure can be applied to the node-based pattern (Figure 7c) connecting the hexagon centers which was probably first mentioned by Ventrella [56] and its design is better for point indexing than the vertex one. The derivation of exact mathematical properties of the Gosper fractal is crucial for the Node-Gosper SFC definition and we briefly remind them here. See papers [7,8,57,58] for details.

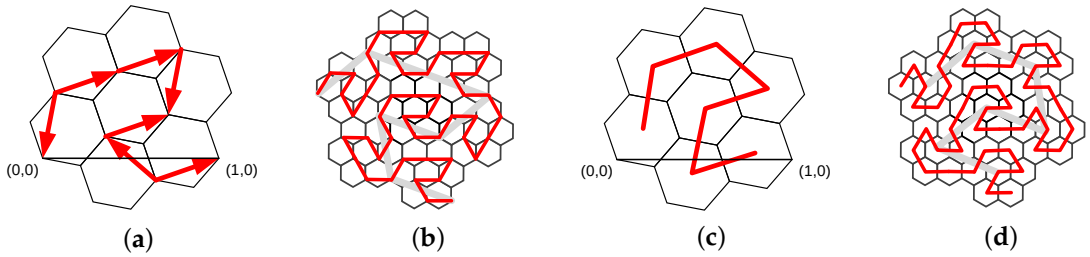


Figure 7. The basic patterns of the Gosper curve: (**a**) 7-pattern of vertex representation, (**b**) the second iteration of vertex representation, (**c**) 7-pattern of node representation, (**d**) the second iteration of node representation.

Figure 8 describes the transformation of the pattern replacing the base hexagon inscribed in the unit circle of $size = 1$ (black dashes) by seven smaller hexagons (level-1 Gosper island). A hexagon itself practically represents a level-0 Gosper island. The figure also shows two features of the Gosper islands: The Gosper island preserves the area of the parent hexagon, but its perimeter limit is infinity. The scale factor $r = \frac{1}{\sqrt{7}}$ can be computed according to the blue triangle (Figure 8) using the Law of Cosine and it also represents the size of child hexagons. The angle $\alpha = \arcsin\left(\frac{\sqrt{3}}{2\sqrt{7}}\right)$ can be computed using the Law of Sine and the scale factor r. Figure 7a shows that the patterns replacing differently oriented arrows have to be additionally rotated. The angles of orientation are defined by function $f(y)$ in (1). Going through the basic Gosper pattern (Figure 7a) from the first vertex $(0, 0)$ to the last vertex $(1, 0)$, the hexagons 0 and 3 are rotated by -120° and the hexagon 5 by 120°. The transformations can be easily computed using the well-known matrices for rotation (2), scale and translation.

$$f(y) = \begin{cases} -120^\circ & \text{if } y \in \{0, 3\} \\ +120^\circ & \text{if } y = 5 \\ 0^\circ & \text{else} \end{cases} \tag{1}$$

$$R_\gamma = \begin{bmatrix} \cos\gamma & -\sin\gamma \\ \sin\gamma & \cos\gamma \end{bmatrix} \tag{2}$$

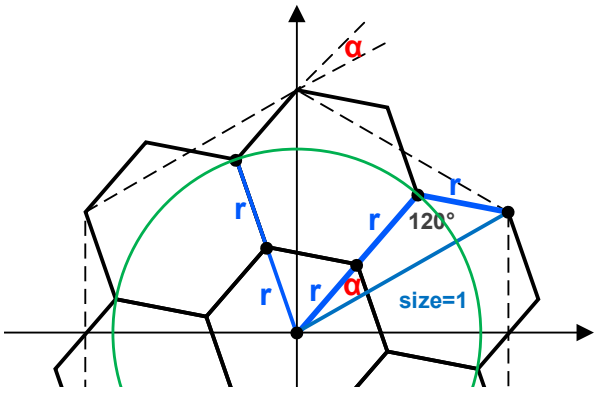


Figure 8. Grid transformation for Gosper islands construction.

3.2. Node-Gosper Indexation

While the recursive method described in the previous section serves only to draw an image of the Gosper fractal with geometrical continuity, the indexation model must precisely state the order of the clusters along the SFC. Each point $\vec{p}$ of the point cloud is localized in the hexagonal grid and

the corresponding hexagon is assigned a bit hash code representing its serial number. This section explains our indexation model of the hexagons ordered by the Node-Gosper SFC.

The basic patterns defining rotation and indexation are shown in Figure 9. The main point is that the subordinate patterns have different order of indices depending on the indexation of the parent pattern. For the indexation model, the geometrical continuity is insufficient.

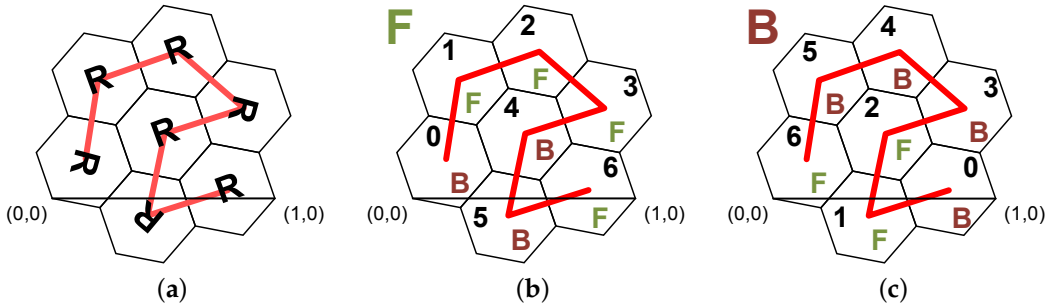


Figure 9. Indexation and passage order of the Node-Gosper pattern. Figure (**a**) shows the basic rotation of subordinate patterns. The other two figures represent the patterns with (**b**) forward (F) and (**c**) backward (B) indexation.

Thus, two structural patterns are described in Figure 9b,c. They define both forward (F) and backward (B) indexing variants. The F-pattern is initial and the curve indexation is calculated in the top-down manner. Each hexagon is recursively replaced by a corresponding subordinate pattern depending on the designation of its parent. Notice that the F/B designation in the backward indexation is just the opposite direction to the forward indexation. This secures the smooth passage of all hexagons along the curve without any large leaps and crossings. The final hierarchical indexation based on the patterns from Figure 9 is displayed in Figure 10. A hash code consists of $n \in \mathbb{N}\backslash\{0\}$ indices $\vec{k} = (k_1, k_2, \ldots, k_n)$, where $k_i \in \mathbb{N}$, addressing the Gosper islands containing a query point on each level $i \in \{1, 2, \ldots, n\}$ of the hexagonal hierarchy. The indices of the upper hierarchical levels are represented by more significant trinities and the lower ones by less significant trinities. The hash code is computed as $code = k_1 \cdot 2^{3\cdot(n-1)} + k_2 \cdot 2^{3\cdot(n-2)} + \cdots + k_{n-1} \cdot 2^{3\cdot 1} + k_n \cdot 2^{3\cdot 0}$. Inversely, an index k_i can be obtained from *code* by simple bit operations. An index k_i is a number from 0 to 6, but the three bits can represent maximally the number 7. Thus, there is unfortunately a numerical gap breaking the requirement of a contiguous sequence of indices (codes) stated in Section 2.2. However, it does not have to be a crucial problem. While having a 64-bit unsigned integer, a 21-level hexagonal indexation can be defined.

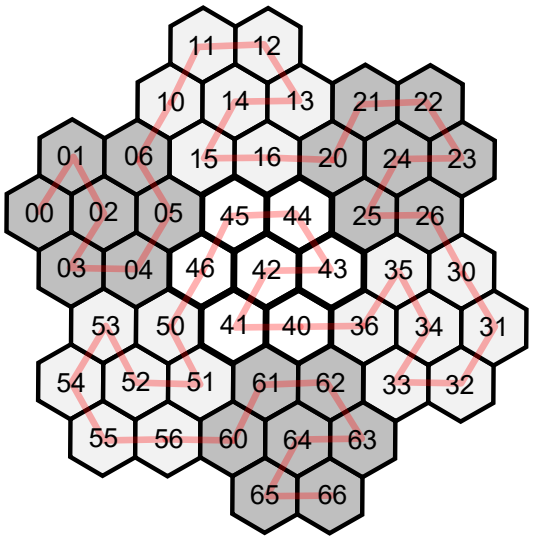


Figure 10. Indexation of the level-2 Node-Gosper curve.

3.3. Localization with Node-Gosper Curves

As emphasized before, the hexagons are not hierarchically decomposable. However, the Gosper fractal (Section 3.1) and Node-Gosper indexation (Section 3.2) are defined recursively in the top-down manner. At first, we define a top-down algorithm computing a centroid of hexagon hierarchically addressed by indices $\vec{k}$. In the next section, we use this method to define the inverse algorithm mapping the points onto the Node-Gosper SFC.

Let us consider the case of the base hexagon with $size = 1$ (Figure 11), which is recursively replaced by the Node-Gosper patterns (Figure 9) up to the level n. Given a vector of indices $\vec{k} = (k_1, k_2, \ldots, k_{n-1}, k_n)$ addressing a hexagon in a level-n Gosper island, a center point of such a hexagon is computed. This algorithm represents the backward mapping which verifies the bijectivity of the Gosper mapping.

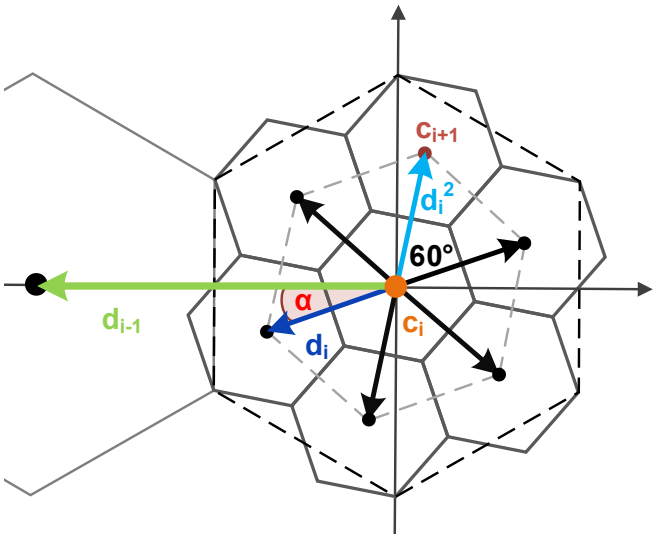


Figure 11. The transformation of directional $\vec{d}_i$ and position $\vec{c}_i$ vectors.

The subordinate patterns can be indexed in the forward (F) or backward (B) manner (Figure 9), which affects the addressing order of the hexagons by $\vec{k}$. A pattern passing order F/B of the i-th recursion can be determined depending on $\vec{k}$ as follows:

$$Order(i) = \begin{cases} F & \text{if } i = 1 \\ B & \text{if } i > 1 \text{ and } Order(i-1) = F \\ & \quad \text{and } k_{i-1} \in \{0, 4, 5\} \\ B & \text{if } i > 1 \text{ and } Order(i-1) = B \\ & \quad \text{and } k_{i-1} \notin \{1, 2, 6\} \\ F & \text{otherwise} \end{cases} \tag{3}$$

Each k_i index has to be recalculated to index $k_i^{'} \in \mathbb{N}$ in the F order (Figure 9) to unify addressing of the corresponding Gosper islands in the top-down manner. The calculation of $\vec{k}^{'} = (k_1^{'}, k_2^{'}, \ldots, k_{n-1}^{'}, k_n^{'})$ is defined as follows:

$$k_i^{'} = \begin{cases} 6 - k_i & \text{if } Order(i) = B \\ k_i & \text{otherwise} \end{cases} \tag{4}$$

For example, a point $\vec{p}$ hashed onto indices $\vec{k}_p = (5, 1)$ according to the level-2 Node-Gosper curve (Figure 10) belongs to the hexagon 5 ($Order(2) = B$) on the first hexagonal level ($Order(1) = F$), whose subordinate island thus has reverse indexation of hexagons. Therefore, the corresponding index

is 1 on the second level of recursion and not 5, as it would be according to the F order, which means $\vec{k}'_p = (5,5)$. A pattern rotation is considered in the following transformations.

$\vec{k}'$ is used to compute the center point of the addressed hexagon. The algorithm recursively transforms two vectors: Position vector $\vec{c}_i$ and directional vector $\vec{d}_i$ (see Figure 11), where i is the current level of recursion. A vector $\vec{c}_i$ represents the initial center position of the current Gosper island. The vector $\vec{d}_i$ represents the rotation and scale of the grid on the i-th level of recursion. The initial directional vector is pointing to the neighboring hexagon and it is defined as $\vec{d}_0 = (w,0)$, where w is the width of the base hexagon. The $\vec{d}_0$ defines the basic orientation and distance between two neighboring hexagon centers before the first transformation. Depending on $\vec{k}'$, $\vec{d}_i$ is defined by the recurrent formula:

$$\vec{d}_0 = (w,0), \quad \vec{d}_i = \frac{1}{\sqrt{7}} \cdot R_\alpha \cdot R_{f(k'_{i-1})} \cdot (\vec{d}_{i-1})^T \text{ for } i \in \langle 1,n \rangle, \tag{5}$$

where $\frac{1}{\sqrt{7}} \cdot R_\alpha$ represents the scale by $r = \frac{1}{\sqrt{7}}$ and rotation by the angle $\alpha = \arcsin\left(\frac{\sqrt{3}}{2\sqrt{7}}\right)$ (Section 3.1), and the $R_{f(k'_{i-1})}$ represents the additional rotation defined by (1). The grid transformation can be simply defined by the following matrix:

$$G = \frac{1}{\sqrt{7}} \cdot R_\alpha = \begin{bmatrix} \frac{5}{14} & -\frac{\sqrt{3}}{14} \\ \frac{\sqrt{3}}{14} & \frac{5}{14} \end{bmatrix}. \tag{6}$$

Next, a vector $\vec{d}_i^{\,k'_i}$ pointing to the specific neighboring hexagon addressed by k'_i is defined as:

$$\vec{d}_i^{\,k'_i} = R_{\gamma(k'_i)} \cdot (\vec{d}_i)^T \text{ for } k'_i \in \{0,1,2,3,5,6\}, \quad \vec{d}_i^{\,4} = (0,0), \tag{7}$$

where $\gamma(k'_i)$ represents the corresponding angle by which $\vec{d}_i$ has to be rotated, so that the $\vec{d}_i^{\,k'_i}$ points to the hexagon addressed by the k'_i index. $\gamma(y)$ is defined as:

$$\gamma(y) = \gamma_y, \text{ where } y \in \{0,1,\ldots,6\} \text{ and } \vec{\gamma} = (0^\circ, 60^\circ, 120^\circ, 180^\circ, 0^\circ, -60^\circ, -120^\circ). \tag{8}$$

where $\gamma(y)$ equals to a multiple of 60°. The hexagon with $y = 0$ is initial, and therefore, $\vec{d}_i^{\,0} = \vec{d}_i$. For example, $\vec{d}_i$ is rotated by 120° to obtain the second hexagon center $\vec{d}_i^{\,2}$ (see Figure 11).

The center point $\vec{c}_i$ is computed by a translation of $\vec{c}_{i-1}$ by the properly rotated directional vector $\vec{d}_{i-1}^{\,k'_{i-1}}$ as:

$$\vec{c}_i = \vec{c}_{i-1} + \vec{d}_{i-1}^{\,k'_{i-1}} \text{ for } i \in \langle 2,n \rangle, \quad \vec{c}_1 = (0,0). \tag{9}$$

The center point $\vec{c}_n$ represents the final hexagon center addressed by indices $\vec{k}$.

3.4. Node-Gosper Point Mapping

In the previous section, we described the algorithm computing a hexagon center according to $\vec{k}$. In this section, we define the inverse mapping of a point onto the Node-Gosper curve, so that $\vec{k}$ is computed and the corresponding hash is generated.

The mapping onto the Node-Gosper curve can be done only in the bottom-up manner, because hexagons cannot be decomposed into sub-hexagons (Figure 12). As it was explained in the previous sections, the hexagonal 7-pattern (Gosper island) is rotated and scaled to replace a parent hexagon which defines the required hierarchy of hexagons. The constructed Gosper islands are non-convex fractal tiles, so that they have to be fitted into neighboring islands to tile the plane. The top-down point localization is incorrect, because the jagged edges of islands overstep the borders of parent

hexagons, so that a point previously localized in the hexagon (especially close to the edges) may belong to different branch of hierarchy in the next iteration (Figure 12). An important fact about the hexagonal hierarchy is that seven hexagon centers of each Node-Gosper pattern are always located inside their parent hexagon. It means that the upward hexagon centers localization defines numerically stable inverse procedure to the top-down algorithm and it secures the correct hierarchical assignment regardless of the jagged edges. There are still several problems that have to be solved:

1. The relative position of hexagons of two subsequent grids at level i and $i-1$ is not obvious, because information of the complete parental hierarchy is unknown at level i (Figure 12).
2. The grid rotation varies in the different branches of hierarchy . The specific grid transformation has recursive character and is unknown at level i (Section 3.2).
3. For the same reason, the indexation and passage order are unknown at level i too.

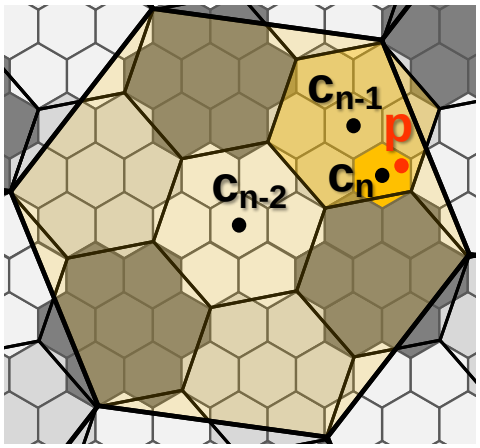


Figure 12. The localization of $\vec{p}$ in level-n Gosper island and the search of the corresponding centers.

To achieve both the correctness of hierarchical localization and recursive principles of the Gosper fractal, we separate the hashing algorithm into two phases. At first, we localize a point $\vec{p}$ in the multi-resolution hexagonal grid in the bottom-up manner using the inverse formulas for the Gosper fractal construction (Figure 12). The hierarchical localization gives us just a vector of indices defining the relative location of a point $\vec{p}$ in the multi-resolution grid. This step secures the correct localization inside the fractal shape of Gosper islands and we describe the procedure in detail in Section 3.4.1. As the Gosper fractal is defined recursively, the specific pattern transformation and indexation for the different recursive branches can be computed only in the top-down manner. Thus, the indices returned by the first part of the algorithm have to be recursively recalculated in the top-down manner (see Section 3.4.2). The second part of the algorithm returns the correct indices that are used to compute the final hash code defining the point order along the Node-Gosper SFC.

3.4.1. Gosper Hierarchical Localization

The task is to determine the relative position of the current hexagon in the context of the parent hexagon. We ignore the difficulties with rotation and passage order in different branches of recursion for start. First, a point $\vec{p}$ is localized in the level-n hexagonal grid (details in [8,43,52]) using the matrix (12) which returns the axial coordinates $\vec{a}_n \in \mathbb{Z}^2$ of the hexagon containing the point $\vec{p}$. The next procedure computes only the relative transformation between axial coordinates which is independent on the specific *size* of hexagons. As mentioned before, the center points of hexagons can be reliably localized inside the parent hexagon. Thus, our transformation between levels i and $i-1$ consists of three steps:

1. Calculation of hexagon center $\vec{c}_i$ according to axial coordinates $\vec{a}_i$.

2. Transformation of the $\vec{c}_i$ according to the orientation of the parent grid.
3. Localization of the transformed center point in the parent grid and calculation of the new axial coordinates $\vec{a}_{i-1}$.

The superior grid of level-$(i-1)$ should be scaled by $\sqrt{7}$ and rotated by $-\alpha$. However, it is easier to preserve the same transformation matrix for calculation of all indices and alternatively transform the center point (scaled by $\frac{1}{\sqrt{7}}$, rotated by $+\alpha$), so the matrix G (6) is used instead. The transformation between levels i and $i-1$ can be written as follows:

$$T = C^{-1} \cdot G \cdot C, \tag{10}$$

where C is the matrix (11) computing the center of a hexagon according to the axial coordinates, C^{-1} is the inverse matrix (12) computing the axial coordinates from the center point and G is the forward transformation matrix (6) (read [8,43,52] for details).

$$C = \begin{bmatrix} \sqrt{3} & \sqrt{3}/2 \\ 0 & 3/2 \end{bmatrix}. \tag{11}$$

$$C^{-1} = \begin{bmatrix} \sqrt{3}/3 & -1/3 \\ 0 & 2/3 \end{bmatrix}. \tag{12}$$

The final constant matrix T is calculated as:

$$T = \begin{bmatrix} \sqrt{3}/3 & -1/3 \\ 0 & 2/3 \end{bmatrix} \cdot \begin{bmatrix} 5/14 & -\sqrt{3}/14 \\ \sqrt{3}/14 & 5/14 \end{bmatrix} \cdot \begin{bmatrix} \sqrt{3} & \sqrt{3}/2 \\ 0 & 3/2 \end{bmatrix} = \begin{bmatrix} 2/7 & -1/7 \\ 1/7 & 3/7 \end{bmatrix}. \tag{13}$$

$\vec{a}_i$ and $\vec{a}_{i-1}$ are the integer axial coordinates of hierarchically corresponding hexagons at level i and $i-1$. As the transformation matrix T produces real unrounded coordinates, we denote the parent real axial coordinates as $\vec{a}^{\mathbb{R}}_{i-1} \in \mathbb{R}^2$ that have to be rounded to the closest integers $\vec{a}_{i-1} \in \mathbb{Z}^2$. Their hierarchical relationship with child axial coordinates $\vec{a}_i \in \mathbb{Z}^2$ can be derived as:

$$\vec{a}^{\mathbb{R}}_{i-1} = T \cdot (\vec{a}_i)^T,\ \vec{a}_{i-1} = round(\vec{a}^{\mathbb{R}}_{i-1}) \text{ for } i \geq 1. \tag{14}$$

The Equation (14) computes the transformed axial coordinates, but we also need to compute the hexagon index corresponding to the Node-Gosper pattern described in Section 3.2. Let the parent axial coordinates denote as $\vec{a}^{\mathbb{R}}_{i-1} = (q^{\mathbb{R}}, l^{\mathbb{R}})$. They are recalculated to the cube coordinates as $x^{\mathbb{R}} = q^{\mathbb{R}}, z^{\mathbb{R}} = l^{\mathbb{R}}, y^{\mathbb{R}} = -x^{\mathbb{R}} - z^{\mathbb{R}}$ according to the condition $x^{\mathbb{R}} + y^{\mathbb{R}} + z^{\mathbb{R}} = 0$. The decimal part of coordinates represents the relative position between the child and parent hexagon. The relative decimal coordinates $x_d \in \langle -1, 1\rangle$, $y_d \in \langle -1, 1\rangle$, $z_d \in \langle -1, 1\rangle$ are computed as:

$$x_d = x^{\mathbb{R}} - round(x^{\mathbb{R}}), \quad y_d = y^{\mathbb{R}} - round(y^{\mathbb{R}}), \quad z_d = z^{\mathbb{R}} - round(z^{\mathbb{R}}). \tag{15}$$

As each hexagon center is close to one vertex and edge of the parent grid (see Figure 12), the dominant axis $dominant = max\{abs(x_d), abs(y_d), abs(z_d)\}$ and its sign determine the index of the child hexagon in the view of the parent hexagon. The advantage is that this numeric transfer takes into account the negative coordinates of points, so that the dataset can be positioned anywhere in the space. The relative

decimal coordinates $\vec{v}_d = (x_d, y_d, z_d)$ can be recalculated to the index $b_i = B(\vec{v}_d)$ of the default pattern (Figure 13 P1), where $B(\vec{v}_d)$ is defined as:

$$B(\vec{v}_d) = \begin{cases} 0 & \text{if } \forall o \in \{x_d, y_d, z_d\} : abs(o) < Thd \\ 2 & \text{if } x_d \text{ is dominant and } x_d < 0 \\ 5 & \text{if } x_d \text{ is dominant and } x_d > 0 \\ 4 & \text{if } y_d \text{ is dominant and } y_d < 0 \\ 1 & \text{if } y_d \text{ is dominant and } y_d > 0 \\ 6 & \text{if } z_d \text{ is dominant and } z_d < 0 \\ 3 & \text{if } z_d \text{ is dominant and } z_d > 0 \end{cases} \tag{16}$$

The threshold constant $Thd \to 0$ represents some small decimal number ($Thd = 10^{-5}$ works well). As the coordinate transformation works on the level of a unitary grid, the method is numerically stable. The optimized algorithm is summarized in Algorithm 1.

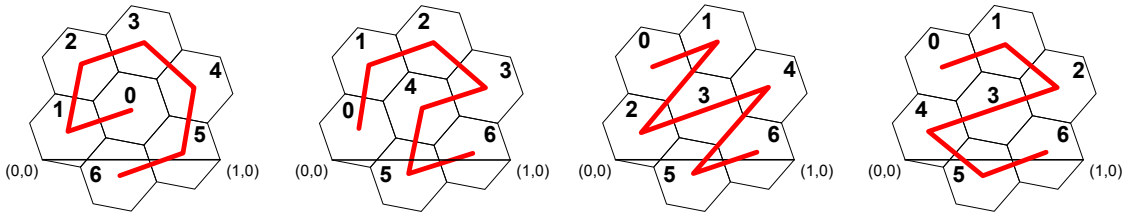


Figure 13. Four different hexagonal indexation patterns. The pattern (P1) is the default one.

Algorithm 1 Hierarchical point mapping

Require:

$\vec{v} = (x, y, z)$: cube coordinates of input point $\vec{p}$

$B(v_d)$: transfers relative coordinates v_d to index b_i of the default pattern (16)

T: transformation matrix (13)

n: number of hierarchical levels

Ensure:

returns the indices $\vec{k} = (k_0, \ldots, k_n)$ of a point $\vec{p}$

{coordinate transformation from cube $\vec{v}$ to axial $\vec{a}$}

$\vec{a}.q = \vec{v}.x$, $\vec{a}.l = \vec{v}.z$

{for all levels}

for i $\leftarrow 1$ **to** n **do**

{transformation of axial coordinates $\vec{a}$}

$\vec{a} = T \cdot (\vec{a})^T$

{coordinate transformation from axial $\vec{a}$ to cube $\vec{v}$}

$\vec{v}.x = \vec{a}.q$, $\vec{v}.z = \vec{a}.l$, $\vec{v}.y = -\vec{v}.x - \vec{v}.z$

$\vec{a}.q = (INT)round(\vec{a}.q)$, $\vec{a}.l = (INT)round(\vec{a}.l)$

{decimal part $\vec{v}_d$ computation}

$\vec{v}_d.x = \vec{v}.x - (INT)round(\vec{v}.x)$

$\vec{v}_d.y = \vec{v}.y - (INT)round(\vec{v}.y)$

$\vec{v}_d.z = \vec{v}.z - (INT)round(\vec{v}.z)$

{transfer to final index k_i}

$k_{n-i} = B(v_d)$

end for

3.4.2. Index Conversion

Section 3.4.1 described a general method for hierarchical localization of a point $\vec{p}$ in the level-n Gosper island. This section shows how to recalculate the default indexation model to other models. Especially, it defines a method solving the difficulties like pattern rotation and passage order to construct the correct Node-Gosper SFC.

Let $\vec{b} = (b_1, \ldots, b_n)$ be a vector of indices returned by Algorithm 1, where n is the number of hierarchical levels. Each $b_i \in \langle 0, 6 \rangle$ can be further transfered to final index $k_i = E(t, b_i)$, where $E(t, b_i)$ is the function defined in Table 1. The different indexation patterns are shown in Figure 13 and their conversion is very simple. The final vector of indices $\vec{k} = (k_1, \ldots, k_n)$ defines the hierarchical address of a hexagon in the Gosper island to define the final order of hexagons. Each k_i can be represented by three bits, so all the indices are masked to the corresponding bits of hash according to the index significance (read Section 3.2).

Table 1. Table defining the transfer function $E(t, b_i)$ between default index b_i of pattern (P1) and patterns (P2), (P3) and (P4) according to types in Figure 13, where $t \in \{\text{P1, P2, P3, P4}\}$.

$E(t, b_i)$ / b_i	**0**	**1**	**2**	**3**	**4**	**5**	**6**
$t = P1$	0	1	2	3	4	5	6
$t = P2$	4	0	1	2	3	6	5
$t = P3$	3	2	0	1	4	6	5
$t = P4$	3	4	0	1	2	6	5

The procedure computing the correct Node-Gosper curve is more complicated. There are two tasks:

1. The indices have to be hierarchically recalculated according to the rotation specified by patterns at different levels of recursion (see Figure 9a).
2. The order of hexagons has to be hierarchically changed according to the forward and backward indexation (see Figure 9b,c).

These properties have recursive nature and they cannot be solved by bottom-up localization. The $\vec{b}$ has to be recalculated to obtain the indexation model described in Section 3.2. The whole method is described in Algorithm 2. Here, we summarize it in two steps:

1. As the patterns are rotated only by $\pm 120^\circ$ (1), the rotation can be done by index shifting. The default indexation has the index 0 on the central hexagon and the indices 1-6 form a ring of hexagons. Let the $rotDir_i \in \{-1, 0, 1\}$ be a number symbolizing the pattern rotation on the i-th level by $\{-120^\circ, 0^\circ, +120^\circ\}$. The shifted index is computed as explained in Algorithm 2 (lines 2–11).
2. The passage order is reversed depending on the current pattern as explained in Section 3.3. The algorithm is described in Algorithm 2 (lines 12–16).

Algorithm 2 Node-Gosper index computation

Require:
$\vec{b}$: vector of default indices of input point $\vec{p}$
$E(t, b_i)$: index transfer function defined in Table 1
$order = F$: passage order
$rotDir = 0$: rotation of the pattern ($rotDir \in \{-1, 0, 1\}$)
n: number of hierarchical levels
Ensure:
returns the indices $\vec{k} = (k_1, \ldots, k_n)$ of a point $\vec{p}$

{for all levels}
for i $\leftarrow 1$ **to** n **do**
{rotation: shift of indices}
  if $b_i \neq 0$ **and** $rotDir \neq 0$ **then**
    $b_i = b_i + 2 \cdot rotDir$
    if $b_i < 1$ **then** $b_i = b_i + 6$
    else if $b_i > 6$ **then** $b_i = b_i - 6$
    end if
  end if
{transfer to Node-Gosper basic pattern}
  $k_i^0 = E(P2, b_i)$
{rotation calculation for next level}
  if $k_i^0 \in \{0, 3\}$ **and** $(--rotDir < -1)$ **then** $rotDir = 1$
  else if $k_i^0 = 5$ **and** $(++rotDir > 1)$ **then** $rotDir = -1$
  end if
{index order reversion}
  if $order = B$ **then** $k_i = 6 - k_i^0$
  else $k_i = k_i^0$
  end if
{pattern passage order determination}
  if $k_i^0 \in \{0, 4, 5\}$ **then** $order = (order = B\,?\,F : B)$
  end if
end for

4. Experiments and Discussion

This section evaluates our novel Node-Gosper SFC based on the hexagonal grid defined in Section 3. Subsection 4.1 compares the complexity and distance metrics of our algorithm with older methods and SFCs. Subsection 4.2 evaluates the pros and cons of the presented curve.

4.1. Comparison

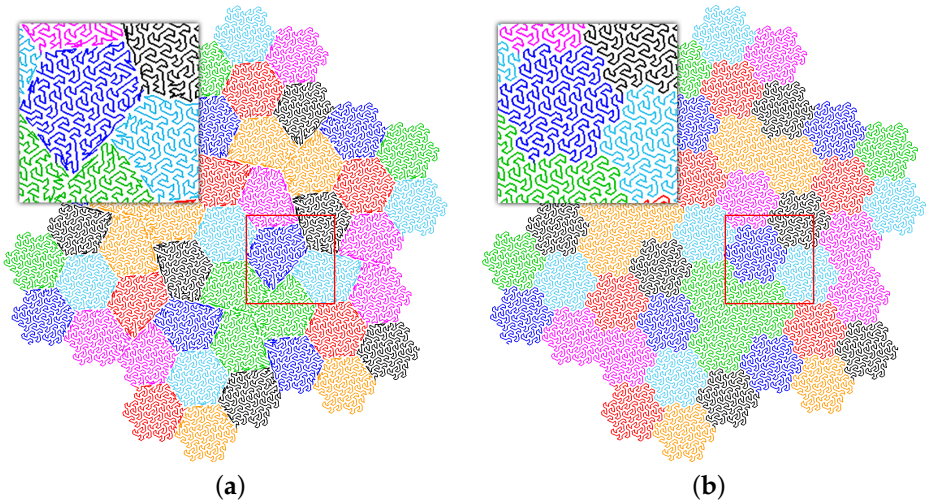
Our method is compared with the older approximative top-down Gosper SFC [7], the naive bottom-up approach [8] and several widely used triangular and orthogonal SFCs. After extensive research, we have not found any other Gosper-based SFC method for comparison. The section is divided into three subsections comparing correctness of point localization, execution time and distance metrics.

All experiments run on the following hardware: Intel Core i7-7700HQ @ 2.8 GHz, 16 GB RAM, Windows 10 64-bit

4.1.1. Jagged Edges

Incorrect hierarchical point localization during the construction of the Node-Gosper SFC induces wrong point mapping along the curve [7]. The general Gosper islands are non-convex fractal tiles, so that the jagged edges of islands overstep the borders of parent hexagons. The previous algorithms [7,8] could not efficiently deal with jagged edges (Figure 14). The experiment presented in Figure 14 uses a point cloud representing the nodes of the level-5 Node-Gosper curve that are hashed

according to the top-down and bottom-up methods. The figure confirms that the correct Node-Gosper SFC cannot be constructed by top-down decomposition of hexagons [7], because it produces different types of polygonal clusters that do not preserve the features of the hexagonal grids. The misclassified points are accumulated close to the solid hexagonal edges between the Gosper islands (Figure 14A). Our method uses the bottom-up approach (Figure 14B) composing the Gosper islands from subordinate hexagons, which secures right point localization in the hexagonal hierarchy and produces the correct Node-Gosper SFC passing through the nodes of a regular hexagonal grid. The Gosper islands are undamaged and respecting the jagged edges (Figure 14B).



(**a**) (**b**)

Figure 14. The comparison of (**a**) top-down and(**b**) bottom-up level-5 Node-Gosper SFC. The top-down method obviously cannot solve the problem with jagged edges, because it produces non-hexagonal sub-clusters.

4.1.2. Efficiency

The efficiency of the curve construction is also a crucial problem, because an SFC with expensive hashing function is not suitable for practical usage regardless of the outstanding theoretical metrics. Next experiment is conducted to compare the execution times of our new optimized Node-Gosper method with other algorithms including the naive implementation of Node-Gosper curve [8] and the Node-Gosper simple curve, which uses our new mapping algorithm (Algorithm 1) without additional index conversion (second recursive part) required for the correct Node-Gosper SFC. The simple variant is faster and can be used for general hierarchical hexagonal model without quality requirements on the final SFC. As the complexity of a hashing function is considered to be constant, the construction times are measured using a randomly generated dataset of 10^6 uniformly distributed points. The chart in Figure 15 shows that our new algorithm is four times faster than the naive old one. In rough comparison with non-hexagonal SFCs (Z-order, Hilbert, Sierpińsky), the Node-Gosper hashing algorithm is approximately two times slower. However, this is a logical consequence of applied tiling. Unlike with quads (4-pattern) or triangles (2-pattern), seven sub-clusters have to be considered at each level, thus the number of clusters generated by Node-Gosper curve is much greater for the same hierarchical level than the number of quads, e.g., for $n = 8$ a complete SFC contains $2^8 = 256$ triangles, $4^8 = 65536$ quads and $7^8 = 5764801$ hexagons. It means that the hashing function of Node-Gosper curve is more time consuming but the construction time covers much greater number of sub-clusters creating finer tessellation of the space. To compare SFCs based on different grids, we compute the total construction time per cluster:

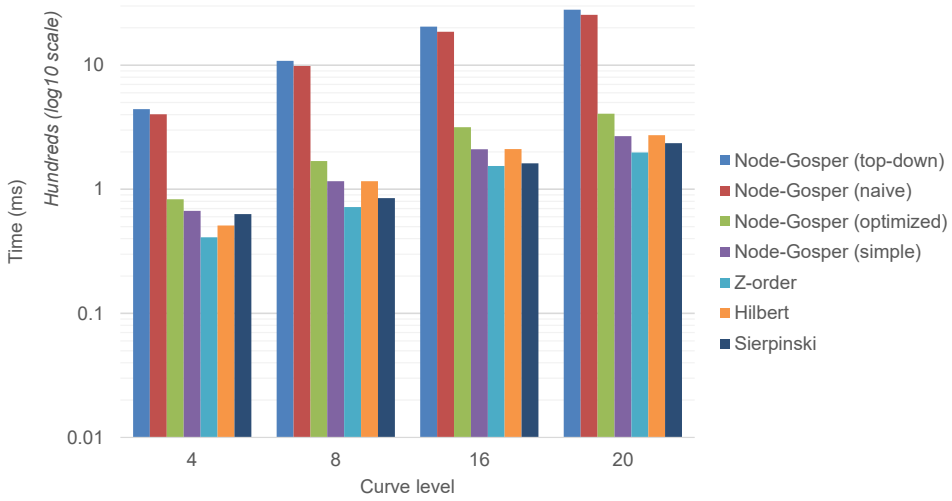


Figure 15. The graph shows a comparison of construction times of our Node-Gosper (optimized) method with selected space-filling curves including the previous naive Node-Gosper algorithm [8], top-down Node-Gosper algorithm [7] and simple Node-Gosper (Figure 13 P2) algorithm grouping the hexagons into Gosper islands without additional pattern rotation and clusters reordering. Each measurement was performed on a random dataset with 10^6 points for different levels of clustering.

Definition 1. *Let time be the total construction time of the SFC, let level be the hierarchical level until which the SFC is generated and let clusters be the number of sub-clusters of the tiling pattern, the time per cluster is computed as:*

$$clusterTime = \frac{time}{clusters \cdot level} \tag{17}$$

The metric in Definition 1 takes into account the number of clusters that have to be checked during the evaluation of hashing function which is applicable for any SFC based on the different types of grids. This metric is computed and compared for all SFCs in the chart in Figure 16. The chart shows that our optimized Node-Gosper SFC is actually faster than other good SFCs such as Hilbert and Sierpińsky curve and way faster than older variants of the Node-Gosper SFC. Moreover, the tiling with Gosper islands has Arrwwid number $\alpha = 3$ [42], which means that a sufficiently small disc hits at most 3 tiles at each level, while $\alpha = 4$ for orthogonal and $\alpha = 6$ for triangular grids. This feature can be successfully utilized to define an efficient query structure based on the Node-Gosper curve in the future.

4.1.3. Distance Metrics

The last test is focused on the distance metrics of SFCs. The basic requirement for a good SFC is that two points along a curve should be close in the multidimensional space as well. The principal metric here is the sum of distances between the query point and 64 closest points along the curve (32 antecedent, 32 following). The metric average of all the points in the dataset is displayed in the charts in Figure 17. The charts compare four variants of the Node-Gosper curve (patterns Figure 13 P1–P4) and other 3 commonly used SFCs tested with six different datasets summarized in Table 2. The first three ones are randomly generated datasets with uniform or Gaussian distribution. The Gaussian islands is a dataset containing several clusters with Gaussian distribution. The datasets S1, S2 and S3 were downloaded from the web page [59]. Experiments show that the Node-Gosper curve is comparable with Hilbert or Sierpiński curves, its metrics are stable according to the standard deviation and its much better than other simple variants of SFCs. The top-down variant seems to have comparable metrics, but points are not localized correctly along the curve, so that it is not a good option.

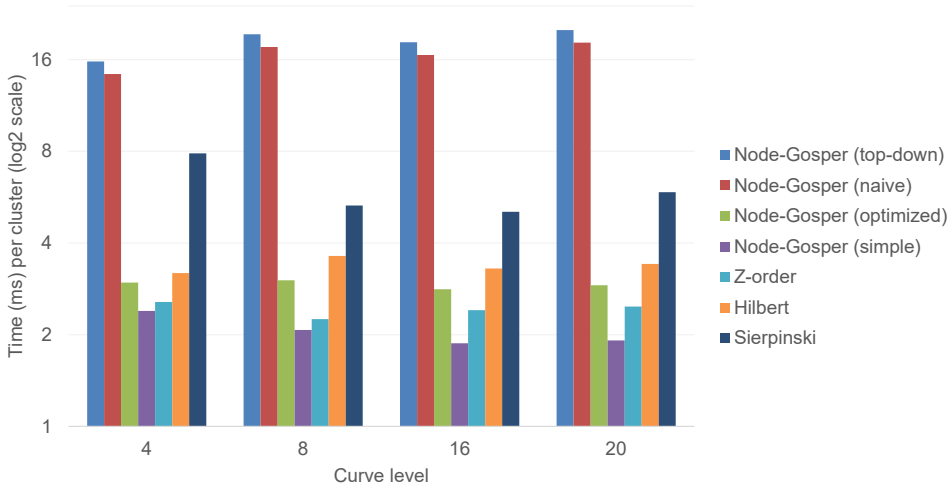


Figure 16. The graph shows total construction times per cluster. Each method has a different number of sub-clusters per pattern (triangular Sierpińsky: 2, orthogonal Z-order and Hilbert: 4, hexagonal Node-Gosper: 7), and thus, produces different number of clusters on the same hierarchical level. This metric is computed as (total time/(sub-clusters × curve level)) and it helps to compare the real time complexity of different point indexing methods. The graph compares of our novel Node-Gosper (optimized) method with the older ones [7,8] and other SFCs. Each measurement was performed on a random dataset with 10^6 points for different levels of clustering.

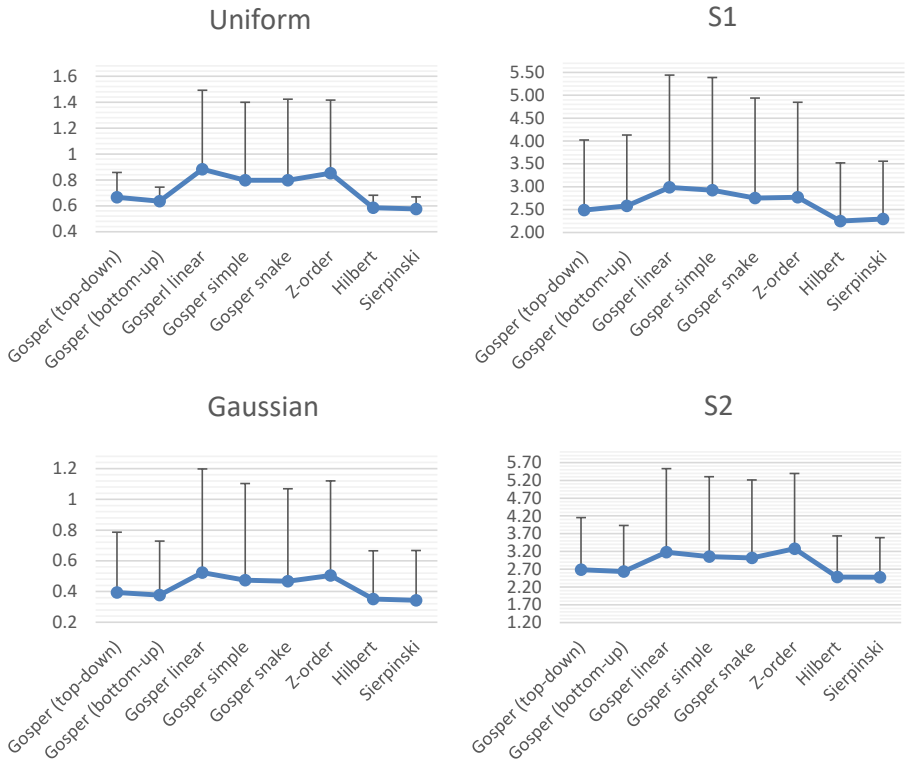


Figure 17. *Cont.*

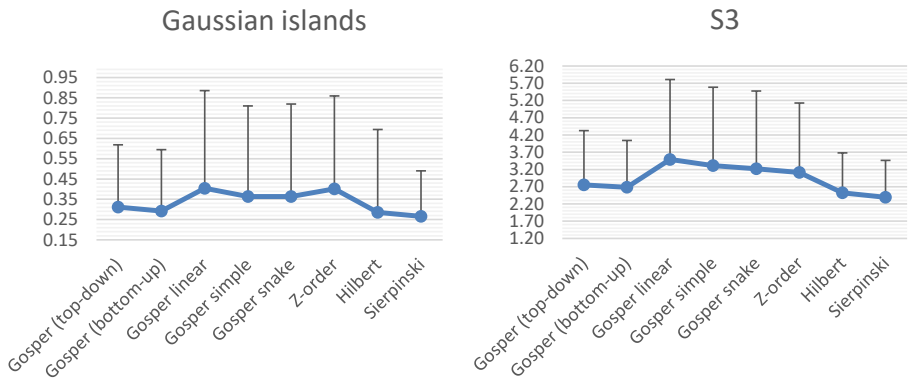


Figure 17. The comparison of distance metrics of different curves. For each point the sum of distances to the closest 64 neighbors lying along the curve represents the principal metric value. The graph shows average and standard deviation of values of all points from a dataset. The distances were normalized by length of the bounding box diagonal. The Gosper SFCs represent the Node-Gosper type of the curve.

Table 2. The list of tested datasets with corresponding point number and the length of the bounding box diagonal. The datasets were obtained from website [59].

Dataset	Point Num. ($\|V\|$)	Diagonal Length
Uniform random	10^5	1.41
Gauss. random	10^5	0.77
Gauss. random islands	10^5	1.28
S1	5000	1,301,821.12
S2	5000	1,345,787.39
S3	5000	1,260,858.57

4.2. Properties

Finally, we summarize the main properties of our proposed curve which fulfill the requirements for a good SFC.

1. Bijective mapping—Sections 3.2–3.4 defined how to map a point onto the Node-Gosper curve and how to compute a cluster point according to the given code.
2. The curve is space-filling—every point of space lies on the curve. The dataset has to be fitted into the circle inscribed into the Gosper island.
3. Total ordering—there are no pattern crossings, thus each cluster is visited by SFC only once.
4. Simplicity and efficiency—the designed algorithms are easy and efficient to compute according to Figures 15 and 16.
5. Localization— points close on the curve are also spatially close (see Figure 17).
6. Sequentialisation—the Node-Gosper SFC does not lead to continuous sequence of code numbers.

The only trouble here is the sequentialisation, because our SFC is based on the 7-pattern, so there is a code gap between the cluster levels. However, it does not have to be a real problem for general clustering purposes. Sparse datasets lead to an incomplete SFC anyway, because the empty areas make an indexation gap along the curve, so most of the clustering methods do not rely on it. Overall, the Node-Gosper curve is a very good hexagonal SFC.

5. Conclusions

The main contribution of this paper is the new hexagonal space-filling curve (SFC) algorithm called the Node-Gosper SFC based on the recursive Gosper fractal. To the best of our knowledge, there is no paper defining a Gosper-based point indexing method except our previous naive proposals. As the multi-resolution hexagonal grids are highly required in various scientific fields such as computer

graphics, image processing, data visualization and geographic information systems (GISs), our hexagonal Node-Gosper SFC represents an effective point indexing method applicable in those areas. This paper summarizes the background about the hierarchical hexagonal point clustering and the corresponding theory. It shows that the multi-resolution hexagonal grids have several outstanding properties including uniform distance between cells and the best coverage of circular queries. The paper surveys the existing SFC methods and highlights the current problems of the hexagonal clustering including the issues with hierarchical decomposition of hexagons, fractal shape of hexagonal composites and their reasonable indexation. Our method solves all those issues. In comparison with our older approaches, the novel method can deal with jagged edges and it is four times faster. The tests showed that the curve has similar distance properties as widely preferred non-hexagonal SFCs (Hilbert, Sierpinski, etc.) and its construction is even faster. Thus, our Node-Gosper SFC represents a scalable, rapid and stable method for hexagonal hierarchical indexing with good quality parameters.

The future work aims at hierarchical hexagonal structures construction and their application for the practical tasks including data visualization or location indexing in GISs. The discussed metrics and measurements already bring us promising results in areas like nearest neighbors search or range query.

Author Contributions: Conceptualization, V.U. and P.G.; methodology, V.U.; software, V.U.; validation, V.U., M.R. and Y.-C.L.; formal analysis, V.U. and P.G.; investigation, V.U.; resources, V.U., V.S. and Y.-C.L.; data curation, V.U., P.G. and V.S.; writing–original draft preparation, V.U.; writing–review and editing, P.G., V.S., M.R. and Y.-C.L.; visualization, V.U.; supervision, V.S. and Y.-C.L.; project administration, V.U. and V.S.; funding acquisition, V.S.

Funding: This work was supported by SGS project, VSB-Technical University of Ostrava, under the grant no. SP2019/141.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

GIS	Geographic Information System
NN	Nearest Neighbor
PC	Point Cloud
SFC	Space-Filling Curve

References

1. Bader, M. *Space-Filling Curves: An Introduction with Applications in Scientific Computing*; Springer Publishing Company, Incorporated: Heidelberg/Berlin, Germany, 2012.
2. Lawder, J.K.; King, P.J.H. Using Space-Filling Curves for Multi-dimensional Indexing. In *Proceedings of the 17th British National Conferenc on Databases: Advances in Databases*; Springer-Verlag: London, UK, 2000; pp. 20–35.
3. Skopal, T.; Krátký, M.; Pokorný, J.; Snášel, V. A New Range Query Algorithm for Universal B-trees. *Inf. Syst.* **2006**, *31*, 489–511.
4. Lauterbach, C.; Garland, M.; Sengupta, S.; Luebke, D.; Manocha, D. Fast BVH Construction on GPUs. *Comput. Graph. Forum* **2009**, *28*, 375–384.
5. Isaac, T.; Burstedde, C.; Ghattas, O. Low-Cost Parallel Algorithms for 2:1 Octree Balance. In Proceedings of the 2012 IEEE 26th International Parallel Distributed Processing Symposium (IPDPS), Shanghai, China, 21–25 May 2012; pp. 426–437.
6. Gardner, M. Mathematical Games—In which "monster" curves force redefinition of the word "curve". *Sci. Am.* **1976**, *235*, 124–133.
7. Uher, V.; Gajdoš, P.; Radecký, M.; Snášel, V. A proposal of hierarchical vertex clustering based on the Gosper curve. In Proceedings of the 2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Budapest, Hungary, 9–12 October 2016; pp. 632–637.

8. Uher, V.; Gajdoš, P.; Snášel, V. Towards the Gosper Space Filling Curve Implementation. In Proceedings of the 2017 3rd IEEE International Conference on Cybernetics (CYBCONF), Exeter, UK, 21–23 June 2017; pp. 1–8.
9. Rusu, R.B.; Cousins, S. 3D is here: Point Cloud Library (PCL). In Proceedings of the 2011 IEEE International Conference on Robotics and Automation, Shanghai, China, 9–13 May 2011; pp. 1–4.
10. Samet, H. *Foundations of Multidimensional and Metric Data Structures*; Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA, 2005.
11. Brodsky, I. H3: Uber's Hexagonal Hierarchical Spatial Index. 2019. Available online: https://eng.uber.com/h3/ (accessed on 20 April 2019).
12. Gajdoš, P.; Ježowicz, T.; Uher, V.; Dohnálek, P. A parallel Fruchterman–Reingold algorithm optimized for fast visualization of large graphs and swarms of data. *Swarm Evol. Comput.* **2016**. *26*, 56–63.
13. Uher, V.; Gajdoš, P.; Ježowicz, T. Solving nearest neighbors problem on GPU to speed up the Fruchterman-Reingold graph layout algorithm. In Proceedings of the 2015 IEEE 2nd International Conference on Cybernetics (CYBCONF), Gdynia, Poland, 24–26 June 2015; pp. 305–310.
14. Pharr, M.; Jakob, W.; Humphreys, G. *Physically Based Rendering: From Theory to Implementation*; Morgan Kaufmann: Burlington, MA, USA, 2016.
15. Langetepe, E.; Zachmann, G. *Geometric Data Structures for Computer Graphics*; AK Peters, Ltd.: Natick, MA, USA, 2006.
16. Sonka, M.; Hlaváč, V.; Boyle, R. *Image Processing, Analysis, and Machine Vision*; Cengage Learning: Boston, MA, USA, 2014.
17. Ahuja, N. On approaches to polygonal decomposition for hierarchical image representation. *Comput. Vis. Graph. Image Process.* **1983**, *24*, 200–214.
18. Middleton, L.; Sivaswamy, J. *Hexagonal Image Processing: A practical Approach*; Advances in Computer Vision and Pattern Recognitio; Springer: London, UK, 2006.
19. Sahr, K. Hexagonal discrete global grid systems for geospatial computing. *Arch. Photogramm. Cartogr. Remote Sens.* **2011**, *22*, 363–376.
20. Ben, J.; Tong, X.; Chen, R. A spatial indexing method for the hexagon discrete global grid system. In Proceedings of the 2010 18th International Conference on Geoinformatics, Beijing, China, 18–20 June 2010; pp. 1–5.
21. Mahdavi-Amiri, A.; Harrison, E.; Samavati, F. Hexagonal connectivity maps for Digital Earth. *Int. J. Digit. Earth* **2015**, *8*, 750–769.
22. Mahdavi-Amiri, A.; Samavati, F.; Peterson, P. Categorization and Conversions for Indexing Methods of Discrete Global Grid Systems. *ISPRS Int. J. Geo-Inf.* **2015**, *4*, 320–336.
23. Mahdavi-Amiri, A.; Harrison, E.; Samavati, F. Hierarchical Grid Conversion. *Comput. Aided Des.* **2016**, *79*, 12–26.
24. Wang, D.; Xu, L.; Peng, J.; Robila, S. Subdividing Hexagon-Clustered Wireless Sensor Networks for Power-Efficiency. In Proceedings of the 2009 WRI International Conference on Communications and Mobile Computing, Kunming, China, 6–8 January 2009; Volume 2, pp. 454–458.
25. Bandara, H.M.N.D.; Jayasumana, A.P.; Illangasekare, T.H. A Top-Down Clustering and Cluster-Tree-Based Routing Scheme for Wireless Sensor Networks. *Int. J. Distrib. Sens. Netw.* **2011**, *7*, 1–17.
26. Márquez, A.; Ángel Plaza.; Suárez, J.P. Hamiltonian triangular refinements and space-filling curves. *J. Comput. Appl. Math.* **2019**, *346*, 18–25.
27. Su, T.; Wang, W.; Lv, Z.; Wu, W.; Li, X. Rapid Delaunay triangulation for randomly distributed point cloud data using adaptive Hilbert curve. *Comput. Graph.* **2016**, *54*, 65–74.
28. Ivrissimtzis, I.; Dodgson, N.; Sabin, M. A generative classification of mesh refinement rules with lattice transformations. *Comput. Aided Geom. Des.* **2004**, *21*, 99–109.
29. Alexa, M. Refinement operators for triangle meshes. *Comput. Aided Geom. Des.* **2002**, *19*, 169–172.
30. Franklin, W.R. Nearest Point Query on 184M Points in E3 with a Uniform Grid. In Proceedings of the 17th Canadian Conference on Computational Geometry (CCCG'05), Windsor, ON, Canada, 10–12 August 2005; pp. 239–242.

31. Vitter, J.S. External Memory Algorithms and Data Structures: Dealing with Massive Data. *ACM Comput. Surv. (CSUR)* **2001**, *33*, 209–271.
32. Butz, A.R. Convergence with Hilbert's Space Filling Curve. *J. Comput. Syst. Sci.* **1969**, *3*, 128–146.
33. Lam, W.M.; Shapiro, J.M. A Class of Fast Algorithms for the Peano-Hilbert Space-Filling Curve. In Proceedings of the 1st International Conference on Image Processing, Austin, TX, USA, 13–16 November 1994; Volume 1, pp. 638–641.
34. Breinholt, G.; Schierz, C. Algorithm 781: Generating Hilbert's Space-Filling Curve by Recursion. *ACM Trans. Math. Softw.* **1998**, *24*, 184–189.
35. Middleton, L.; Sivaswamy, J. Edge detection in a hexagonal-image processing framework. *Image Vis. Comput.* **2001**, *19*, 1071–1081.
36. Ohn, S.Y. Neighborhood Decomposition of Convex Structuring Elements for Mathematical Morphology on Hexagonal Grid. In *Proceedings of the 21st International Conference on Computer and Information Sciences*; Springer-Verlag: Berlin/Heidelberg, Germnay, 2006; pp. 511–521.
37. Bell, S.; Diaz, B.; Holroyd, F.; Jackson, M. Spatially referenced methods of processing raster and vector data. *Image Vis. Comput.* **1983**, *1*, 211–220.
38. Burt, P.J. Tree and pyramid structures for coding hexagonally sampled binary images. *Comput. Graph. Image Process.* **1980**, *14*, 271–280.
39. Gibson, L.; Lucas, D. Vectorization of raster images using hierarchical methods. *Comput. Graph. Image Process.* **1982**, *20*, 82–89.
40. Vince, A. Indexing the aperture 3 hexagonal discrete global grid. *J. Vis. Commun. Image Represent.* **2006**, *17*, 1227–1236.
41. Sahr, K. On the Optimal Representation of Vector Location using Fixed-Width Multi-Precision Quantizers. *ISPRS-Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci.* **2013**, *XL-4/W2*, 1–8.
42. Haverkort, H.J. Recursive tilings and space-filling curves with little fragmentation. *J. Comput. Geom.* **2011**, *2*, 92–127.
43. Uher, V.; Gajdoš, P.; Ježowicz, T.; Snášel, V. Application of Hexagonal Coordinate Systems for Searching the K-NN in 2D Space. In *Innovations in Bio-Inspired Computing and Applications*; Advances in Intelligent Systems and Computing; Springer International Publishing: Berlin/Heidelberg, Germany, 2016; Volume 424, pp. 209–220.
44. Muelder, C.; Ma, K.L. Rapid Graph Layout Using Space Filling Curves. *IEEE Trans. Vis. Comput. Graph.* **2008**, *14*, 1301–1308.
45. Auber, D.; Huet, C.; Lambert, A.; Renoust, B.; Sallaberry, A.; Saulnier, A. GosperMap: Using a Gosper Curve for Laying Out Hierarchical Data. *Vis. Comput. Graph. IEEE Trans.* **2013**, *19*, 1820–1832.
46. Wyvill, B. Painting with Flowsnakes. In *Proceedings of the Workshop on Computational Aesthetics*; Eurographics Association: Aire-la-Ville, Switzerland, 2015; pp. 171–182.
47. Hales, T.C. The honeycomb conjecture. *Discret. Comput. Geom.* **2001**, *25*, 1–22.
48. Mandelbrot, B.B. *The Fractal Geometry of Nature*; W. H. Freeman and Co.: New York, NY, USA, 1983.
49. Haverkort, H.; van Walderveen, F. Locality and bounding-box quality of two-dimensional space-filling curves. *Comput. Geom.* **2010**, *43*, 131–147.
50. Blåsjö, V. The Isoperimetric Problem. *Am. Math. Mon.* **2005**, *112*, 526–566.
51. Chang, H.C.; Wang, L.C. A simple proof of Thue's Theorem on circle packing. *arXiv* **2010**, arXiv:1009.4322.
52. Patel, A.J. Red Blob Games—Hexagonal Grids. 2019. Available online: http://www.redblobgames.com/grids/hexagons/ (accessed on 20 March 2019).
53. Bédorf, J.; Gaburov, E.; Portegies Zwart, S. A Sparse Octree Gravitational N-body Code That Runs Entirely on the GPU Processor. *J. Comput. Phys.* **2012**, *231*, 2825–2839.
54. Sahr, K.; White, D.; Kimerling, A.J. Geodesic Discrete Global Grid Systems. *Cartogr. Geogr. Inf. Sci.* **2003**, *30*, 121–134.
55. Smith, W.D. Space-filling curves, Randomness, and Geometry Problems. 2019. Available online: http://rangevoting.org/SpaceFillCurve.html (accessed on 20 March 2019).
56. Ventrella, J. Brainfilling Curves—The Root 7 Family. 2019. Available online: http://www.fractalcurves.com/Root7.html (accessed on 20 March 2019).

57. Riddle, L. Classic Iterated Function Systems—Flowsnake. 2019. Available online: http://ecademy.agnesscott.edu/~lriddle/ifs/ksnow/flowsnake.htm (accessed on 20 March 2019).
58. Bishop, C.J.; Jones, P.W.; Pemantle, R.; Peres, Y. The Dimension of the Brownian Frontier is Greater Than 1. *J. Funct. Anal.* **1997**, *143*, 309–336.
59. Fränti, P. Clustering Basic Benchmark. 2019. Available online: https://cs.joensuu.fi/sipu/datasets/ (accessed on 20 March 2019).

© 2019 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).