Quality of alternative terrain models

Schouten, E.

Award date:
2012

technische universiteit eindhoven
Department of Mathematics and Computer Science

Master's Thesis

Quality of alternative terrain models

by
Ed Schouten

Supervisor
dr. Herman Haverkort

Eindhoven, March 18, 2012

Abstract

Though most grid-based geographic information systems store a terrain as a uniform tiling of square grid cells in $\mathbb{R}^2$ and cubes in $\mathbb{R}^3$, it is possible to use different cell shapes. This master thesis shall provide both theoretical and practical analysis of the accuracy of commonly used algorithms for terrains when applied to non-conventional terrain models. We shall look at terrains of hexagonal grid cells and bitruncated cubes specifically.

For rotations on grids we observe that our alternative terrain models provide an increase in accuracy, while for least-cost path computation we measure the opposite; grids with square grid cells (*regular grid cells*) outperform grids with hexagonal cells. When measuring the accuracy of flow accumulation algorithms, we see that the results depend on the weight function used; the function used to determine how precipitation is divided over neighbouring grid cells. In general, terrains of square grid cells outperform hexagonal grid cells for weight functions based on differences in elevation of neighbouring grid cells (the slope). For weight functions based on the angle of steepest descent we measure the opposite.

Also, even though hexagonal grid cells cannot be tiled recursively, we shall see it is possible to create efficient algorithms that allow hexagonal grid cells to be used as building blocks for space-filling curves and hierarchical adaptive grid structures. In this thesis we primarily discuss the Gosper curve, but will also provide new space-filling curves that have some useful additional properties. Finally we discuss the use of septuatrees; a modification of quadtrees that uses the tiling of the Gosper curve. We shall see that these techniques can be implemented having the same or similar running time complexities as their conventional counterparts.

Contents

1 Introduction

For many decades, computer systems have been used in the area of geography to perform simulations and analysis on existing or proposed physical terrains. These systems are often referred to as *geographic information systems*. The types of computations that are performed on terrains are countless. Examples include:

- Models of pressure areas in the atmosphere can be used to simulate weather in the nearby future.[11]
- Analysis of GSM coverage on a terrain allows telecommunication service providers to deploy new antennae more wisely.[15]
- Simulations of water flow on a terrain can help assessing the dangers of flooding and volcanic eruptions.
- Satellite navigation systems can be used to efficiently travel to a given destination, by land, sea or air.

When creating a geographic information system, one of the most important design decisions is the way terrains are encoded within the system. There are many trade-offs that need to be considered when deciding which encoding, called a *terrain model*, suits the application best. Questions a designer can ask him- or herself include:

- Is the terrain model precise enough? Can it be used to compute the results at a decent level of detail? For example, a weather forecast with a 100 km precision may be sufficient to get an overview of the weather in Russia, but this is likely not sufficient to predict weather locally. Especially in mountainous and coastal regions the weather can vary radically across short distances.
- Does the terrain model allow accurate computation? For a consumer-grade satellite navigation system, small errors and detours typically cause little damage in practice, while for simulations of volcano eruptions such mistakes can easily endanger the population of an entire village.
- Does the terrain model allow the computation to be performed in a timely fashion? As computer systems have become affordable, they are now used by mainstream users. Most of these users have little understanding of the actual complexity of a computer system, meaning they typically expect the system to compute results instantaneously.

- Is the terrain model used by the system compatible with existing methods used to acquire the datasets? In other words, can available datasets of terrains be used by the geographical information system with little or no loss of information?

When considering contemporary geographic information systems for two-dimensional spaces, it seems that in practice most often two types of terrain models are used that address these constraints sufficiently, namely *grids* and *triangulated irregular networks* (TINs). Whereas grid models use a raster having a fixed structure to store a certain property of a terrain (e.g. elevation, temperature, pressure), TINs store a terrain as a set of connected triangles, where either the faces, edges or vertices are associated with a value.[12]

Though practically all applications of grids use square grid cells, there is no requirement that this needs to be the case. In this master thesis we shall research the feasibility of using grid cells having non-conventional shapes. In this context, the term feasibility can be interpreted in different ways. First, one can consider feasibility being whether it is in fact possible to implement a fast and robust algorithm that solves a given problem for the terrain model used. Second, feasibility could be used to denote whether this implementation can compute the intended results with enough accuracy.

In this master's thesis we are going to research this feasibility by looking at both aspects. In chapters 3, 4 and 5 we are going to look at the accuracy of several terrain models by analysing the results yielded by the application of well-known algorithms. In the second part of this thesis, we will see whether techniques analogue to Hilbert curves[9] and quadtrees[3] can be used in conjunction with one of our alternative terrain models (hexagonal grid cells) as well.

2 Terrain models used in this document

In this document, we are going to discuss the application of a variety of terrain models. For these terrain models we are going to describe how algorithms have to be adapted to work with these terrain models, but there also will be analysis on how the results generated by these algorithms are affected by this.

As the number of possible terrain models is infinite and diverse, this document will only focus on a small number of terrain models, which are either already used in practice quite often or are not hard to implement. Also, even though parts of the analysis performed in this document can be generalised to any number of dimensions used, we shall restrict ourselves to two-dimensional and three-dimensional spaces.

2.1 Terrain models for $\mathbb{R}^2$

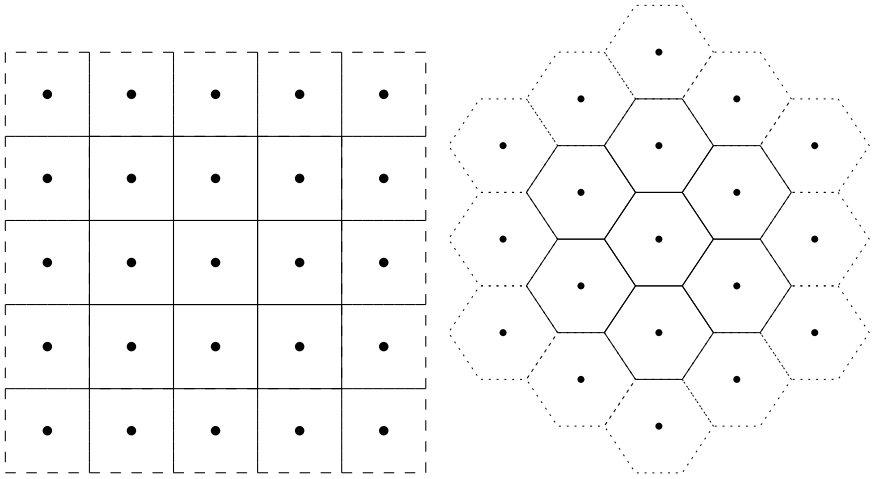
In computational geometry a two-dimensional terrain can be defined as a function, mapping a two-dimensional coordinate on a land surface to a numerical property. Examples of properties of a land surface include pressure, temperature, precipitation and elevation with respect to the sea. Using a two-dimensional terrain to store elevation data, it is possible to recreate a three-dimensional rendering of the shape of the surface. Such terrains are called *digital elevation models*.

The most common way to store a two-dimensional space on a computer system as a grid, is to use regular grid cells. Another nice way to represent grids, is to use hexagonal grid cells, because of their geometric properties, which we shall describe in this section.

2.1.1 Regular grid cells

Probably the easiest and most common way to store a two-dimensional grid terrain on a computer system, is to use regular (rectangular) grid cells, as shown in Figure 2.1(a). What contributes to this fact, is that many programming languages support multi-dimensional arrays as an integral feature. For example, in the C and C++ programming languages it is part of the actual syntax of the language, meaning it is simply a matter of writing `Cell grid[x][y]` to declare a two-dimensional grid of size $x \times y$.

On a regular grid, a grid cell p has eight neighbours (disregarding cells on the boundary of the terrain). Four of these neighbouring grid cells, namely the cardinal neighbours, share two vertices and one edge with p. The other four grid cells, the diagonal neighbours, only share a single vertex with p. Considering a Cartesian grid of regular grid cells where every grid cell has size 1×1, the distance between the centrepoint of a grid cell and its neighbours is either 1 for cardinal neighbours or $\sqrt{2}$ for diagonal neighbours.



(a) Regular grid cells. (b) Hexagonal grid cells.

Figure 2.1: Two-dimensional terrain models used in this document.

This difference in semantics between neighbouring cells has to be taken into account when designing an algorithm that operates on grids. For example, when implementing a shortest path algorithm, diagonal steps have to account for distance $\sqrt{2}$. Since this number is irrational, it is hard – if not impossible – to implement such an algorithm without introducing imprecision. This can be avoided by simply ignoring the diagonal neighbours, but this may lead to suboptimal path calculation.

2.1.2 Hexagonal grid cells

The second two-dimensional terrain model we will be looking at, is a terrain which is tiled with hexagons, instead of rectangles. This model has not seen much use in geographic information systems, but is used in practice in board games and computer games.

Unlike regular grids, where we've seen that the neighbours of a grid cell can be divided in two classes, this is not the case for hexagonal grid cells. Terrains consisting of hexagonal tiles, as shown in Figure 2.1(b), enforce only a single class of neighbours. Each grid cell p has six neighbouring grid cells, all of which share two vertices and one edge with grid cell p. Assuming the hexagons are regular, each neighbouring grid cell has equal distance to p. Unfortunately it is not possible to lay out a grid of regular hexagons on a two-dimensional space using only integer coordinates, which is why we shall use the following two production rules to define how hexagons are placed on the terrain:

- There is a hexagon placed at position $\begin{bmatrix} 0 & 0 \end{bmatrix}$.
- If there is a hexagon placed at position $\begin{bmatrix} x & y \end{bmatrix}$, then it has six neighbouring hexagons: $\begin{bmatrix} x & y+2 \end{bmatrix}$, $\begin{bmatrix} x+1 & y+1 \end{bmatrix}$, $\begin{bmatrix} x+1 & y-1 \end{bmatrix}$, $\begin{bmatrix} x & y-2 \end{bmatrix}$, $\begin{bmatrix} x-1 & y-1 \end{bmatrix}$ and $\begin{bmatrix} x-1 & y+1 \end{bmatrix}$.

This implies that the x-coordinate of a grid cell is even, if and only if its y-coordinate is even as well. Even though this coordinate model allows us to use integer coordinates exclusively, it has the disadvantage that the terrain is stretched in the y-direction by a factor $\sqrt{3}$. This affects the way coordinates on the plane can be rotated. Instead of using the standard rotation matrix $R(\varphi)$, rotation has to be performed as follows:

φ	$\boldsymbol{R}(\varphi)$	$\boldsymbol{R'}(\varphi)$
0	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
$\frac{1}{3}\pi$	$\begin{bmatrix} 1/2 & -\sqrt{3}/2 \\ \sqrt{3}/2 & 1/2 \end{bmatrix}$	$\begin{bmatrix} 1/2 & -1/2 \\ 3/2 & 1/2 \end{bmatrix}$
$\frac{2}{3}\pi$	$\begin{bmatrix} -1/2 & -\sqrt{3}/2 \\ \sqrt{3}/2 & -1/2 \end{bmatrix}$	$\begin{bmatrix} -1/2 & -1/2 \\ 3/2 & -1/2 \end{bmatrix}$
π	$\begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$
$-\frac{2}{3}\pi$	$\begin{bmatrix} -1/2 & \sqrt{3}/2 \\ -\sqrt{3}/2 & -1/2 \end{bmatrix}$	$\begin{bmatrix} -1/2 & 1/2 \\ -3/2 & -1/2 \end{bmatrix}$
$-\frac{1}{3}\pi$	$\begin{bmatrix} 1/2 & \sqrt{3}/2 \\ -\sqrt{3}/2 & 1/2 \end{bmatrix}$	$\begin{bmatrix} 1/2 & 1/2 \\ -3/2 & 1/2 \end{bmatrix}$

Table 2.1: Rotation matrices for standard angles.

$$R'(\varphi) = \begin{bmatrix} 1 & 0 \\ 0 & \sqrt{3} \end{bmatrix} \cdot \begin{bmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1/\sqrt{3} \end{bmatrix} \tag{2.1}$$

$$= \begin{bmatrix} \cos\varphi & -\sin\varphi/\sqrt{3} \\ \sqrt{3}\sin\varphi & \cos\varphi \end{bmatrix} \tag{2.2}$$

Rotation matrices for six standard angles are given in Table 2.1. Notice that even though the rotation matrices contain fractions, they can safely be applied to coordinates on the plane using only integer arithmetic, by performing the division by two only after performing addition. Because odd rows are only combined with odd columns and vice versa, we only divide even numbers by two.

2.2 Terrain models for $\mathbb{R}^3$

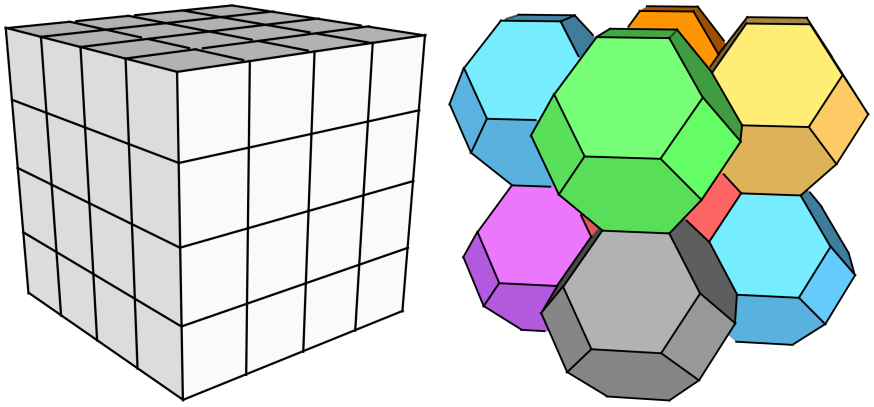
Though two-dimensional terrains are very well suited for modelling land surfaces, there are cases in which a three-dimensional environment needs to be modelled by a computer system to perform a simulation. Examples include simulations of heat dissipation of a combustion engine or air flow simulation of a hull of an airplane. For these kinds of simulations a certain property needs to be stored for any point in $\mathbb{R}^3$ – not just the points of an outer surface.

For the terrain models for $\mathbb{R}^2$ we observed that it is possible to create a layout of grid cells where the distance between centrepoints of neighbouring grid cells is uniform. Unfortunately such a tiling does not exist for $\mathbb{R}^3$.

Theorem 1 *There does not exist a tiling of grid cells for $\mathbb{R}^3$ where the distance between centrepoints of all pairs of neighbouring grid cells is equal.*

Proof. Suppose there exists a tiling of grid cells where the distance between centrepoints is equal. When looking at the dual graph of such a tiling, every maximum set of pairwise adjacent grid cells form the vertices of a polyhedron. As the distance between each pair of vertices of this polyhedron must be the same, the polyhedron can only be a regular tetrahedron. These tetrahedra form a face-to-face tiling of $\mathbb{R}^3$. This causes a contradiction, as it is impossible to tile the three-dimensional space with regular tetrahedra.[16] □

We shall therefore compare two models where these distances are non-uniform, namely cubes with a tiling of bitruncated cubes.



(a) Cubed grid cells. (b) Bitruncated cubed grid cells.

Figure 2.2: Three-dimensional terrain models used in this document.

2.2.1 Cubed grid cells

As the use of cubes as grid cells as shown in Figure 2.2(a) can be considered a logical extension of regular grid cells to $\mathbb{R}^3$, it shares both the advantages and disadvantages of this model. Just like regular grid cells, they can easily be mapped to a multi-dimensional array. When looking at the neighbour relationship, things start to become more complex.

As the number of dimensions increase, so do the number of classes of neighbouring grid cells. A cubed grid cell has 26 neighbours that can be divided in three classes:

- Six face adjacent neighbours having distance 1,
- Twelve edge adjacent diagonal neighbours having distance $\sqrt{2}$, and
- Eight vertex adjacent neighbours having distance $\sqrt{3}$.

This means that the distance between neighbouring grid cells may vary up to $\frac{\sqrt{3}}{1} \approx +73\%$.

2.2.2 Bitruncated cubed grid cells

Another way to tile a three-dimensional space, is by using a grid of bitruncated cubes, as shown in Figure 2.2(b). Bitruncated cubes consist of 14 faces, namely 6 squares and 8 hexagons. They can be obtained by truncating the vertices of a regular cube, until exactly half the volume of the cube remains. The easiest way to imagine how they can be tiled in $\mathbb{R}^3$, is by simply placing them in the same layout as normal cubes. Between each group of eight adjacent cubes, this will yield a hollow space, which has the shape of another bitruncated cube.

The dual graph of a tiling of bitruncated cubes is a tiling of disphenoid tetrahedra. These tetrahedra have two edges of length 2 and four edges of length $\sqrt{3}$. Where we saw a 73% difference in edge length for the dual graph of regular cubes, this difference is only $\frac{2}{\sqrt{3}} \approx +15\%$ for bitruncated cubes. Of the tetrahedra that allow a face-to-face tiling in $\mathbb{R}^3$ described by Sommerville[16, 2], this specific instance has the smallest difference in edge length.

Bitruncated cubes can easily be mapped to an environment where coordinates may only consist of integers, for example by using only coordinates where $x + y + z \mod 2 = 0$.

3 Rotating grids

It is a strong claim that terrains modelled using hexagonal grid cells will lead to higher quality computations than grids using regular grid cells. In fact, it is impossible to prove such a thing, since it entirely depends on the application. For example, an algorithm that traverses a terrain in rectilinear directions exclusively will likely not benefit from a terrain model using hexagonal grid cells. In this section we shall look at a single operation that can be performed on terrains, namely rotation. For this operation we can observe that hexagonal grid cells do have an advantage.

Consider a geographic information system that stores an elevation model for the Earth. Such datasets are typically obtained from various providers. These datasets will not only use different measurement techniques, but also various coordinate models based on different standards. This is because like most celestial objects, the Earth is not a perfect sphere. This means that in most cases, the integration of a new dataset into the pool requires the application of fundamental operations such as stretching, skewing and rotating.

In this chapter we are only looking at rotating a terrain using an angle φ around centrepoint r. Independent of the cell structure, we can perform rotation as given in Algorithm 1. Essentially, it computes each cell of the output by obtaining a set of cells of the input. This set of cells is then used to derive a new value, using interpolation.

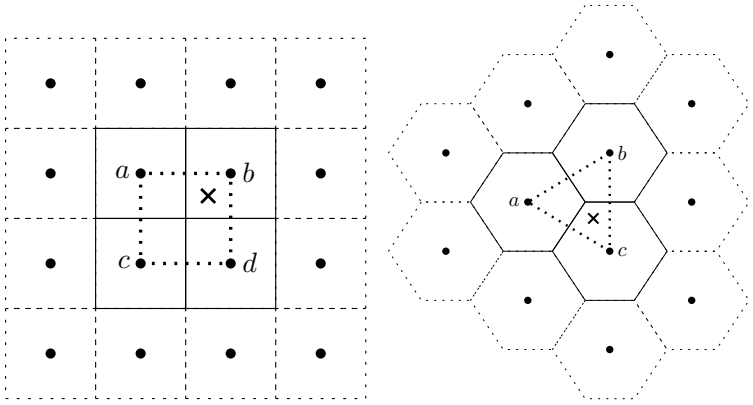
Algorithm 1 ROTATETERRAIN(Terrain in, Terrain out, Point r, Angle φ)

1: **for all** cells $c \in out$ **do**
2: $p_{out} \leftarrow$ coordinate of cell c in terrain out
3: $p_{in} \leftarrow R(-\varphi) \cdot (p_{out} - r) + r$
4: $N \leftarrow$ cells in in close to point p_{in}
5: $c \leftarrow$ INTERPOLATE(p_{in}, N)

It must be noted that this algorithm is intentionally pretty vague. Not only have we omitted what being 'close to point p_{in}' means, we also have not discussed how the interpolation is performed. These two steps are model specific and shall thus be described for regular grid cells and hexagonal grid cells separately.

3.1 Interpolation on regular grid cells

Any point on the grid can always be placed inside or on the boundary of a square formed by four adjacent grid cells, as shown in Figure 3.1(a). Assume for the sake of simplicity the points a to d have coordinates $\begin{bmatrix}0 & 1\end{bmatrix}$, $\begin{bmatrix}1 & 1\end{bmatrix}$, $\begin{bmatrix}0 & 0\end{bmatrix}$ and $\begin{bmatrix}1 & 0\end{bmatrix}$, respectively. Then using bilinear interpolation we can approximate any point within this square as follows:



(a) Regular grid cells. (b) Hexagonal grid cells.

Figure 3.1: Interpolation on two-dimensional terrain models.

$$
\begin{aligned}
V(x,y) &= (1-x)\cdot y \cdot a && (3.1)\\
&+ x \cdot y \cdot b && (3.2)\\
&+ (1-x)\cdot(1-y)\cdot c && (3.3)\\
&+ x \cdot (1-y) \cdot d && (3.4)
\end{aligned}
$$

Now consider the general case where x and y can have arbitrary values. On a regular grid, one can easily use $\lfloor x \rfloor$ and $\lfloor y \rfloor$ to determine inside which square to interpolate. We can then use function V on the remainder of x and y to perform the interpolation.

3.2 Interpolation on hexagonal grid cells

Whereas the bilinear interpolation on regular grid cells uses four neighbouring grid cells, we can use an interpolation scheme for hexagonal grid cells that only depends on three neighbouring grid cells, as shown in Figure 3.1(b), by performing a triangulation.

Assuming there is a triangle placed at points $\begin{bmatrix}0 & 0\end{bmatrix}$, $\begin{bmatrix}1 & 1\end{bmatrix}$ and $\begin{bmatrix}1 & -1\end{bmatrix}$, we have to develop a linear function that has the following properties:

$$
\begin{aligned}
V(0,0) &= a && (3.5)\\
V(1,1) &= b && (3.6)\\
V(1,-1) &= c && (3.7)
\end{aligned}
$$

Because we know function V has the shape $V(x,y) = Ax + By + C$, we can easily derive A, B and C. After simplifying the resulting formula, we obtain:

$$
\begin{aligned}
V(x,y) &= (1-x)\cdot a && (3.8)\\
&+ (\tfrac{1}{2}x + \tfrac{1}{2}y)\cdot b && (3.9)\\
&+ (\tfrac{1}{2}x - \tfrac{1}{2}y)\cdot c && (3.10)
\end{aligned}
$$

Unfortunately we cannot apply this function as easily as on the regular grid. For this model, we cannot use simple flooring of parameters to determine inside which triangle a point is

placed. Instead, we can use a function as given in Algorithm 2. Every unit square on the hexagonal grid is intersected diagonally, with either a positive or a negative slope. This function checks whether the point is above or below this intersection line to determine inside which triangle the point is placed.

Algorithm 2 INWHICHTRIANGLE(Point p)

1: $cx \leftarrow \lfloor p.x \rfloor$
2: $rx \leftarrow p.x - cx$
3: $cy \leftarrow \lfloor p.y \rfloor$
4: $ry \leftarrow p.y - cy$
5: **if** $cx \mod 2 = cy \mod 2$ **then**
6: **if** $rx < ry$ **then**
7: **return** $\left\{ \begin{bmatrix} cx \\ cy \end{bmatrix}, \begin{bmatrix} cx \\ cy+2 \end{bmatrix}, \begin{bmatrix} cx+1 \\ cy+1 \end{bmatrix} \right\}$
8: **else**
9: **return** $\left\{ \begin{bmatrix} cx \\ cy \end{bmatrix}, \begin{bmatrix} cx+1 \\ cy+1 \end{bmatrix}, \begin{bmatrix} cx+1 \\ cy-1 \end{bmatrix} \right\}$
10: **else**
11: **if** $rx < 1 - ry$ **then**
12: **return** $\left\{ \begin{bmatrix} cx \\ cy+1 \end{bmatrix}, \begin{bmatrix} cx \\ cy-1 \end{bmatrix}, \begin{bmatrix} cx+1 \\ cy \end{bmatrix} \right\}$
13: **else**
14: **return** $\left\{ \begin{bmatrix} cx \\ cy+1 \end{bmatrix}, \begin{bmatrix} cx+1 \\ cy+2 \end{bmatrix}, \begin{bmatrix} cx+1 \\ cy \end{bmatrix} \right\}$

This function can return triangles which either point to the left (see Figure 3.1(b)) or to the right. This means that a function similar to V must be developed to interpolate on right-pointing triangles.

3.3 Quality measurement for rotation

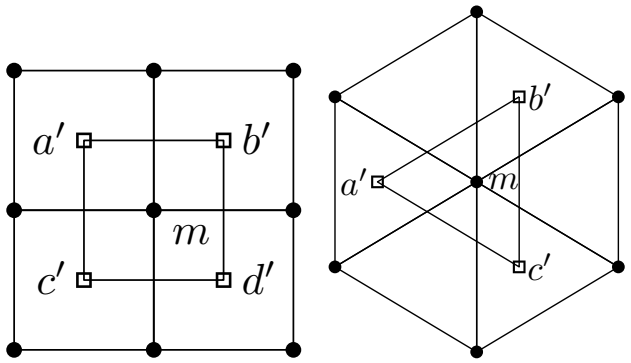
Now that we have described forms of interpolation that can be used to implement a rotation algorithm for terrains, we shall perform a comparison against them to see which of them preserves the data of the terrain better.

Assume we rotate terrain T around centrepoint r, using angle φ, to obtain terrain T'. Now we apply a rotation to terrain T' around the same centrepoint r, but using angle $-\varphi$, to obtain terrain T''. Terrains T and T'' should now encode the same terrain, but the latter will contain various amounts of jitter induced by the rotation operations. All sample points in T'' are partially based on their corresponding sample points in T, but also partially based on cells nearby.

In the remainder of this section, we shall give a theoretical analysis on the ratios of jitter induced by rotation and the results of tests performed on realistic terrains.

3.3.1 Theoretical bounds for preservation of sampled data

Let us define a function $I(a, b')$ that denotes the *ratio of influence* point $a \in T$ has on the value of point $b' \in T'$. It is the multiplication factor that is used by an interpolation function on point a to compute the value of b'. Any interpolated value $V(p')$, where $p' \in T'$ can now be expressed as follows:



(a) Regular grid cells, where $W = \frac{1}{4}$.

(b) Hexagonal grid cells, where $W = \frac{1}{3}$.

Figure 3.2: Overlapping terrains.

$$V(p') = \sum_{a \in T} I(a, p') \cdot V(a) \tag{3.11}$$

We can now apply this mechanism a second time, to obtain an equation for interpolated values of points in terrain T'':

$$V(p'') = \sum_{a' \in T'} \sum_{b \in T} I(a', p'') \cdot I(b, a') \cdot V(b) \tag{3.12}$$

We now define the *worst symmetric ratio of influence* for a specific terrain model as the lowest ratio of influence we can observe between two opposite rotations, for any combination of equal points on the grid and for any placement of T'. Assume there exists a function $M : T \to T''$ that maps sample points on T to the points having the same location in T''. We now define the worst symmetric ratio of influence for a specific terrain model as follows:

$$W = \min_{T'} \min_{p \in T} \sum_{a' \in T'} I(a', M(p)) \cdot I(p, a') \tag{3.13}$$

Using this measurement we can prove the following properties:

Theorem 2 *For the regular grid cell model,* $W \leq \frac{1}{4}$.

Proof. Consider two overlapping terrains of regular grids, as shown in 3.2(a). Assume we have points $m \in T$ and $a', b', c', d' \in T'$. The points in T' are placed right in between four points in T. We can now derive W as follows:

$$\begin{aligned}
W &= \min_{T'} \min_{p \in T} \sum_{a' \in T'} I(a', M(p)) \cdot I(p, a') && (3.14)\\
&\leq \sum_{a' \in T'} I(a', M(m)) \cdot I(m, a') && (3.15)\\
&= I(a', M(m)) \cdot I(m, a') + I(b', M(m)) \cdot I(m, b') && (3.16)\\
&+ I(c', M(m)) \cdot I(m, c') + I(d', M(m)) \cdot I(m, d') && (3.17)\\
&= \tfrac{1}{4} \cdot \tfrac{1}{4} + \tfrac{1}{4} \cdot \tfrac{1}{4} + \tfrac{1}{4} \cdot \tfrac{1}{4} + \tfrac{1}{4} \cdot \tfrac{1}{4} && (3.18)\\
&= \tfrac{1}{4} && (3.19)
\end{aligned}$$

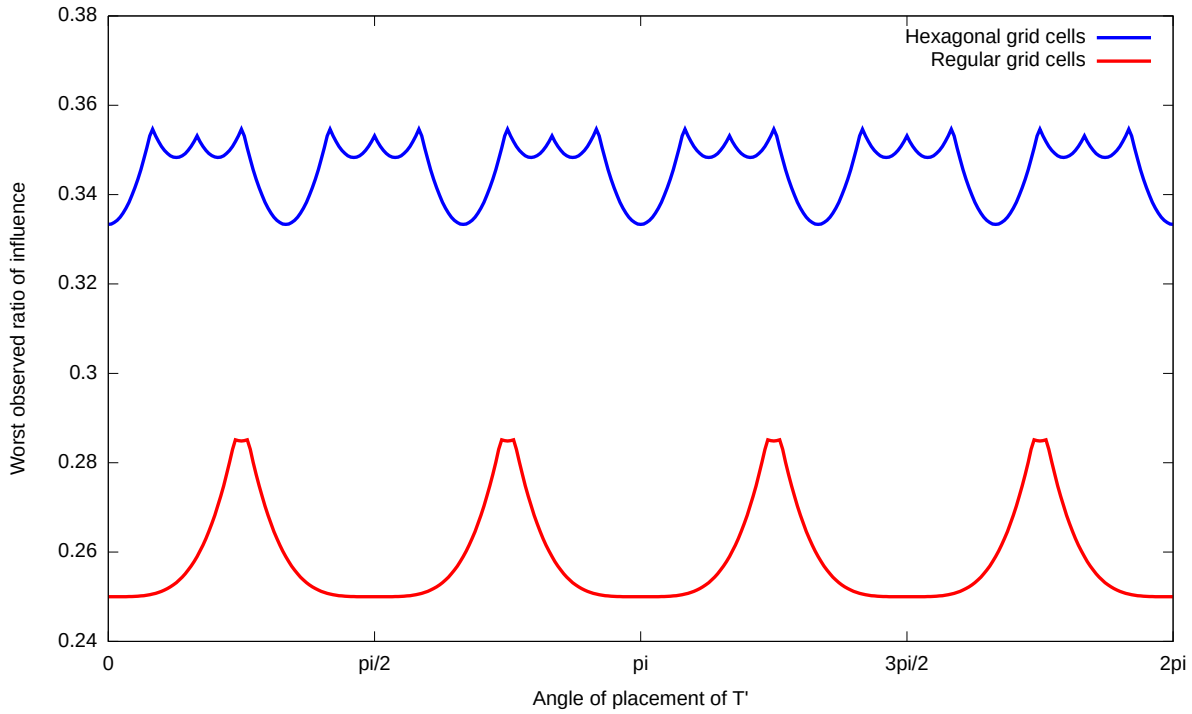


Figure 3.3: Worst observed ratio of influence as a function of the angle of the placement of T'.

□

Theorem 3 *For the hexagonal grid cell model,* $W \leq \frac{1}{3}$.

Proof. The same trick can be repeated for a terrain with hexagonal grid cells, as shown in 3.2(b). Just like for the regular grid, we place the points in T' right in the middle of the interpolation cells of T. This yields the following derivation:

$$
\begin{aligned}
W &= \min_{T'} \min_{p \in T} \sum_{a' \in T'} I(a', M(p)) \cdot I(p, a') && (3.20)\\
&\leq \sum_{a' \in T'} I(a', M(m)) \cdot I(m, a') && (3.21)\\
&= I(a', M(m)) \cdot I(m, a') + I(b', M(m)) \cdot I(m, b') && (3.22)\\
&+ I(c', M(m)) \cdot I(m, c') && (3.23)\\
&= \tfrac{1}{3} \cdot \tfrac{1}{3} + \tfrac{1}{3} \cdot \tfrac{1}{3} + \tfrac{1}{3} \cdot \tfrac{1}{3} && (3.24)\\
&= \tfrac{1}{3} && (3.25)
\end{aligned}
$$

□

Unfortunately, these two theorems have little use in practice. It would have been nicer if we were able to prove a lower bound on W instead of an upper bound. Such a lower bound would act as as a measurement for preservation robustness of sample data. The proofs for the lower bounds are not trivial, since all possible placements of T' must be considered.

Simulations of all possible placements of T' for the regular and the hexagonal grid cell model have been performed, using a resolution of one thousandth the size of a grid cell and rotations in 1° steps. These simulations have not yielded any placements of T' that cause a bound lower

Figure 3.4: The part of the Earth covered by the six GTOPO30 datasets used.

than the upper bounds given previously. This means that it is likely – though unproven – that the upper bounds are in fact tight. Figure 3.3 illustrates the worst observed ratio of influence for all simulated placements observed for a single angle at which T' is placed.

3.3.2 Measurements on realistic datasets

In addition to providing theoretical bounds on the impact of rotation on a grid terrain, we have also performed some practical measurements, using six of the GTOPO30[1] datasets, covering the part of the Earth shown in Figure 3.4. The datasets used have a resolution of 4800×6000 and use a regular grid layout. In order to compare the regular grid layout with the hexagonal layout, the datasets have been downsampled to 1200×1500 and 1290×2791 respectively. Note that both these layouts have approximately the same number of sample points; for the hexagonal grid layout only half of the coordinates is usable.

Rotating back and forth

Together with the original dataset, these two downsampled terrains have been rotated 359 times back and forth around the centrepoint of the grid, using an angle of $0 < \frac{2\pi k}{360} < 2\pi$, where $k \in \mathbb{N}$. Then, for all sample points in all these rotations we have calculated the difference with respect to the original terrain. The average difference and the standard deviation of the difference are shown in Table 3.1.

Rotating iteratively

In addition to performing rotations back and forth, tests have been performed where the terrain is rotated 36 times in a row, using 10° steps. The results are shown in Table 3.2.

[1] `http://eros.usgs.gov/#/Find_Data/Products_and_Data_Available/gtopo30_info`

Sampling	Rect 4800×6000		Rect 1200×1500		Hex 1290×2791	
Dataset	μ	σ	μ	σ	μ	σ
e020n40	10.62	24.31	17.97	40.17	14.17	32.32
e020n90	3.54	7.79	9.68	22.51	8.53	19.87
w020n40	4.07	12.42	8.28	21.17	6.75	17.12
w020n90	9.31	17.39	28.04	49.76	25.18	44.89
w060n40	0.72	1.90	5.26	10.17	4.73	9.17
w060n90	1.02	3.83	5.83	20.65	5.61	19.85

Table 3.1: Mean difference in metres between terrain before and after two rotations, including standard deviation.

Sampling	Rect 4800×6000		Rect 1200×1500		Hex 1290×2791	
Dataset	μ	σ	μ	σ	μ	σ
e020n40	22.13	47.19	34.95	69.64	32.52	66.87
e020n90	8.74	12.28	15.24	16.89	14.53	17.14
w020n40	9.69	25.17	19.98	42.24	18.39	38.80
w020n90	26.40	40.73	46.81	60.50	45.33	61.14
w060n40	5.73	12.83	22.89	32.00	21.31	32.27
w060n90	3.05	14.01	5.78	10.78	6.11	14.31

Table 3.2: Mean difference in metres between terrain before and after 36 ten-degree rotations, including standard deviation.

Observations

As we can observe from these results from the GTOPO30 datasets, performing rotation on a terrain using the hexagonal grid cell model preserves the terrain data slightly better. In almost all cases the mean difference in sample value and its standard deviation are lower.

It must be mentioned though that the rotation performed on these datasets is in reality incorrect. The GTOPO30 datasets encode the map of the Earth as being a rectangle, meaning that any circle of latitude has the same number of sample points. This is not taken into account when performing these rotation benchmarks.

3.4 Extending analysis to $\mathbb{R}^3$

So far we have only considered two-dimensional terrains. The definition of the worst symmetric ratio of influence is, however, dimensionless. We shall now consider three-dimensional spaces.

3.4.1 Cubed grid cells

Since we speculated that the worst symmetric ratio of influence W is equal to $\frac{1}{4}$ for regular grid cells, it may be obvious to assume this bound for cubes in $\mathbb{R}^3$ is equal to $\frac{1}{8}$. Indeed, by placing points in T' in the centre of each of the cubes of T, we do obtain a $\frac{1}{8}$ ratio. Unfortunately, we can also construct instances that are slightly worse.

Theorem 4 *For the cubed grid cell model,* $W \lesssim 0.120436$.

Proof. Assume that the cubes in T, T' and T'' have size $1 \times 1 \times 1$. For terrains T and T'', the cubes are simply axis-aligned and there exists a sample point $m \in T$, having coordinates

Vertex	x	y	z
a'	$\frac{1}{8}(1+\sqrt{5}+\sqrt{2(5-\sqrt{5})})$	$\frac{1}{8}(\sqrt{7-3\sqrt{5}}+\sqrt{5-\sqrt{5}})$	$\frac{1}{16}(5\sqrt{2}+\sqrt{10}-2\sqrt{5-\sqrt{5}})$
b'	$\frac{1}{8}(1+\sqrt{5}-\sqrt{2(5-\sqrt{5})})$	$\frac{1}{16}(5\sqrt{2}+\sqrt{10}+2\sqrt{5-\sqrt{5}})$	$-\frac{1}{16}(-3\sqrt{2}+\sqrt{10}+2\sqrt{5-\sqrt{5}})$
c'	$\frac{1}{8}(1+\sqrt{5}+\sqrt{2(5-\sqrt{5})})$	$-\frac{1}{16}(5\sqrt{2}+\sqrt{10}-2\sqrt{5-\sqrt{5}})$	$-\frac{1}{8}(\sqrt{7-3\sqrt{5}}+\sqrt{5-\sqrt{5}})$
d'	$\frac{1}{8}(1+\sqrt{5}-\sqrt{2(5-\sqrt{5})})$	$\frac{1}{16}(-3\sqrt{2}+\sqrt{10}+2\sqrt{5-\sqrt{5}})$	$-\frac{1}{16}(5\sqrt{2}+\sqrt{10}+2\sqrt{5-\sqrt{5}})$
e'	$-\frac{1}{8}(1+\sqrt{5}-\sqrt{2(5-\sqrt{5})})$	$-\frac{1}{16}(-3\sqrt{2}+\sqrt{10}+2\sqrt{5-\sqrt{5}})$	$\frac{1}{16}(5\sqrt{2}+\sqrt{10}+2\sqrt{5-\sqrt{5}})$
f'	$-\frac{1}{8}(1+\sqrt{5}+\sqrt{2(5-\sqrt{5})})$	$\frac{1}{16}(5\sqrt{2}+\sqrt{10}-2\sqrt{5-\sqrt{5}})$	$\frac{1}{8}(\sqrt{7-3\sqrt{5}}+\sqrt{5-\sqrt{5}})$
g'	$-\frac{1}{8}(1+\sqrt{5}-\sqrt{2(5-\sqrt{5})})$	$-\frac{1}{16}(5\sqrt{2}+\sqrt{10}+2\sqrt{5-\sqrt{5}})$	$\frac{1}{16}(-3\sqrt{2}+\sqrt{10}+2\sqrt{5-\sqrt{5}})$
h'	$-\frac{1}{8}(1+\sqrt{5}+\sqrt{2(5-\sqrt{5})})$	$-\frac{1}{8}(\sqrt{7-3\sqrt{5}}+\sqrt{5-\sqrt{5}})$	$-\frac{1}{16}(5\sqrt{2}+\sqrt{10}-2\sqrt{5-\sqrt{5}})$

Table 3.3: Coordinates of a cube, that shows $W < \frac{1}{8}$.

$\begin{bmatrix}0 & 0 & 0\end{bmatrix}$. We can now place terrain T' in such a way that we enclose point m inside a cube, having the coordinates for its eight vertices, a' to h', as shown in Table 3.3 and Figure 3.5. It is identical to an axis-aligned cube that has been transformed using the rotation matrix $R_x(\frac{\pi}{4}) \cdot R_y(\frac{\pi}{5})$.

As this cube is centered around point $\begin{bmatrix}0 & 0 & 0\end{bmatrix}$, the ratio of influence between these vertices and $M(m)$ is $\frac{1}{8}$ each. The ratio of influence between m and the vertices of the cube can be expressed as $(1-|x|)(1-|y|)(1-|z|)$. This allows us to approximate W as follows:

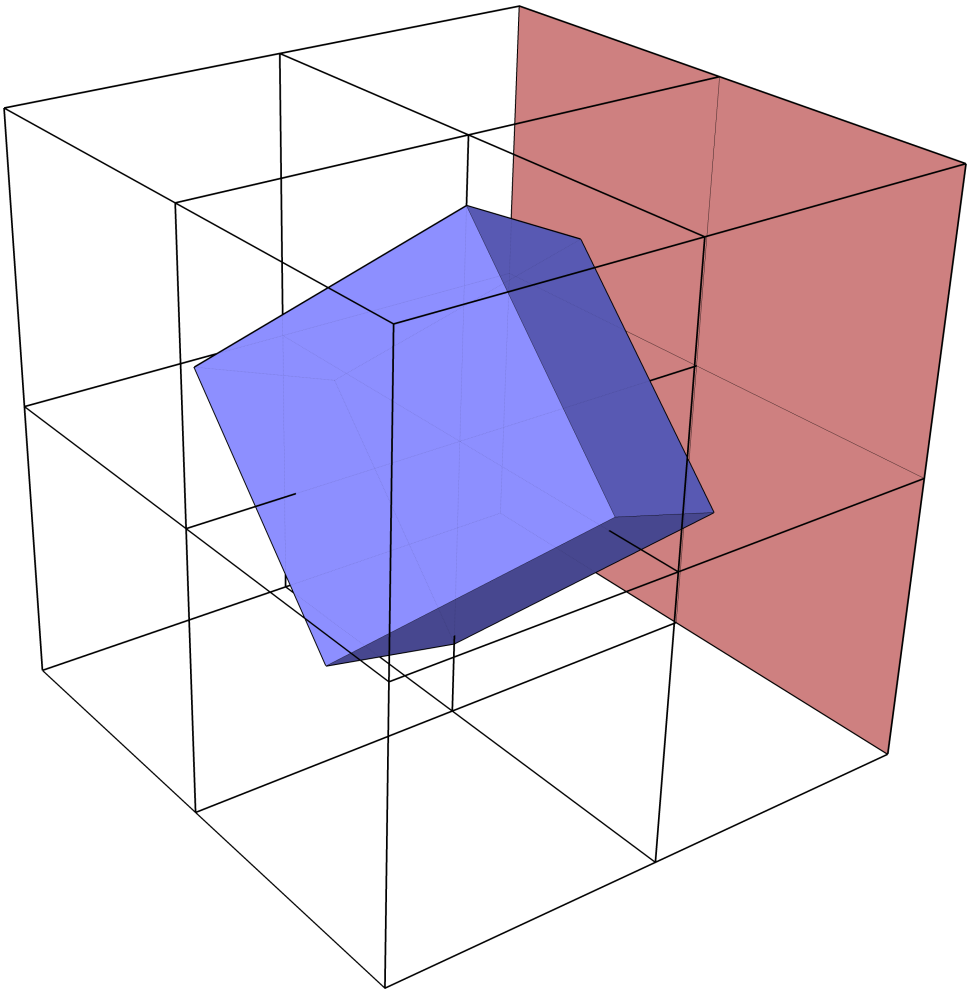
$$
\begin{aligned}
W &= \min_{T'} \min_{p \in T} \sum_{q' \in T'} I(q', M(p)) \cdot I(p, q') && (3.26)\\
&\leq \sum_{q' \in T'} I(q', M(m)) \cdot I(m, q') && (3.27)\\
&= \sum_{q' \in \{a' \ldots h'\}} I(q', M(m)) \cdot I(m, q') && (3.28)\\
&= \sum_{q' \in \{a' \ldots h'\}} \tfrac{1}{8} \cdot (1-|q'.x|)(1-|q'.y|)(1-|q'.z|) && (3.29)\\
&\approx 4 \cdot \tfrac{1}{8} \cdot 0.124191 + 4 \cdot \tfrac{1}{8} \cdot 0.116682 && (3.30)\\
&= 0.120436 && (3.31)
\end{aligned}
$$

Even though it is possible to calculate an exact solution for W, the derivation and its answer are too complex to be of any practical use, hence the approximation. □

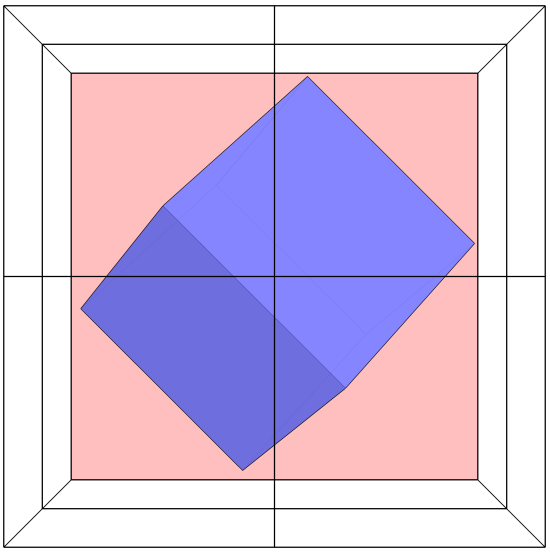
As with the previous terrain models, a simulation of this terrain model has been performed, calculating W as a function of the angle at which the cubes in T' are placed. Because we are working with three-dimensional spaces, cubes have to be rotated in at least two directions to cover all possible placements. In Figure 3.6, angles φ and ψ correspond with rotations around the x and y axes, respectively. These simulations have been used to obtain the placement used in Theorem 4. Not a single placement has been observed that yields a lower upper bound.

3.4.2 Bitruncated cubed grid cells

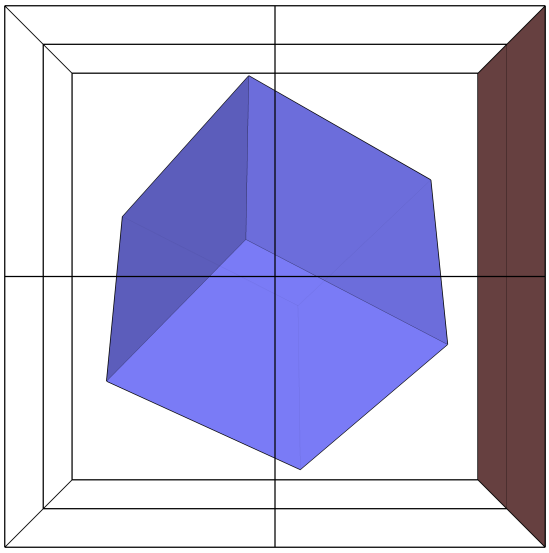
As the dual graph of a tiling of bitruncated cubed grid cells consists of dysphenoid tetrahedra, having edges of length 2 and $\sqrt{3}$, we can develop a linear interpolation scheme as follows. Assume there is a tetrahedron placed at sample points $\begin{bmatrix}0 & 0 & 0\end{bmatrix}$, $\begin{bmatrix}0 & 2 & 0\end{bmatrix}$, $\begin{bmatrix}1 & 1 & 1\end{bmatrix}$ and $\begin{bmatrix}1 & 1 & -1\end{bmatrix}$, having sample values a to d, respectively. Linear interpolation function V can now be defined as:



(a) View from above.



(b) View from the front.



(c) View from the side.

Figure 3.5: 3D renderings of a cube, that shows $W < \frac{1}{8}$, coloured blue. In all images the same surface is coloured red. This colouring has no specific meaning, but eases orientation.

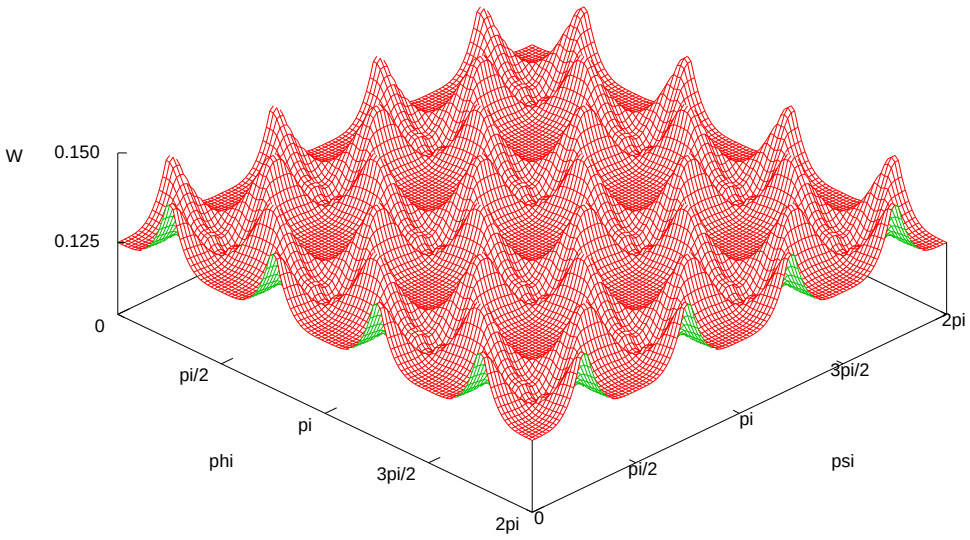


Figure 3.6: Simulation of W for cubes.

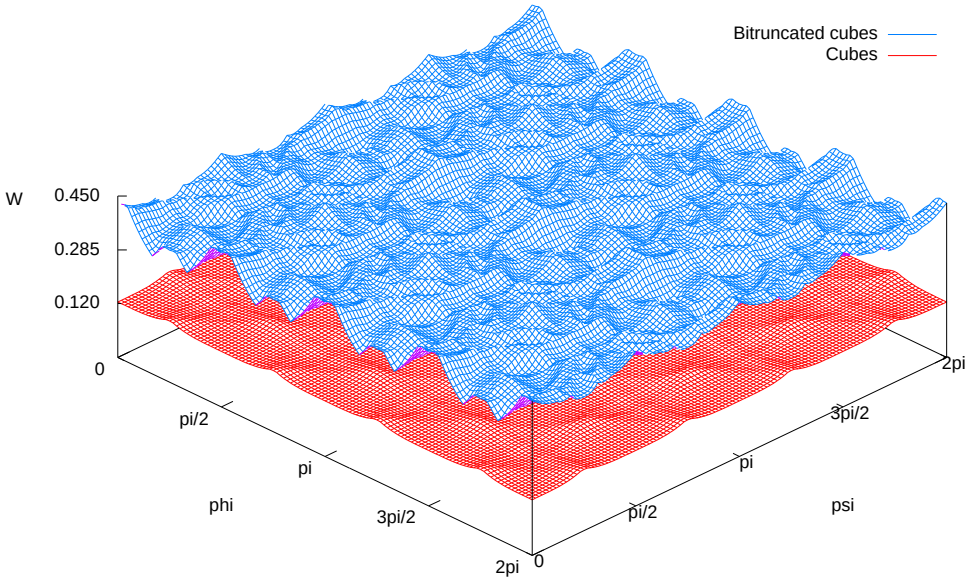


Figure 3.7: Simulation of W for cubes and bitruncated cubes.

Vertex	$\boldsymbol{x}$	$\boldsymbol{y}$	$\boldsymbol{z}$	$\boldsymbol{I(q', M(m))}$	$\boldsymbol{I(m, q')}$
a'	$-\frac{1}{2}$	$\frac{1}{2}\sqrt{3}$	$-\frac{1}{2}$	$\frac{1}{4}$	$\frac{3}{4}-\frac{1}{4}\sqrt{3}$
b'	$-\frac{1}{2}$	$-\frac{1}{2}\sqrt{3}$	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{3}{4}-\frac{1}{4}\sqrt{3}$
c'	$\frac{1}{2}$	$-\frac{1}{2}$	$-\frac{1}{2}\sqrt{3}$	$\frac{1}{4}$	$\frac{3}{4}-\frac{1}{4}\sqrt{3}$
d'	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}\sqrt{3}$	$\frac{1}{4}$	$\frac{3}{4}-\frac{1}{4}\sqrt{3}$

Table 3.4: Placement of a tetrahedron, that shows $W \leq \frac{3}{4}-\frac{1}{4}\sqrt{3}$.

$$\begin{aligned} V(x,y,z) &= (-\tfrac{1}{2}x - \tfrac{1}{2}y + 1) \cdot a && (3.32)\\ &+ (-\tfrac{1}{2}x + \tfrac{1}{2}y) \cdot b && (3.33)\\ &+ (\tfrac{1}{2}x + \tfrac{1}{2}z) \cdot c && (3.34)\\ &+ (\tfrac{1}{2}x - \tfrac{1}{2}z) \cdot d && (3.35) \end{aligned}$$

Even though we managed to prove $W < \frac{1}{8}$ for cubes, this seems to be far from possible for the tiling of bitruncated cubes.

Theorem 5 *For the bitruncated cubed grid cell model,* $W \leq \frac{3}{4} - \frac{1}{4}\sqrt{3} \approx 0.31699$.

Proof. To prove this, we use the same structure as for Theorem 4. Assume we now place a tetrahedron at the coordinates given in Table 3.4. Because the tetrahedron is centered around $\begin{bmatrix}0 & 0 & 0\end{bmatrix}$, the ratio of influence for $M(m)$ is $\frac{1}{4}$. The ratios of influence for a' to d' can be derived by applying V:

$$\begin{aligned} V(x,y,z) &= (-\tfrac{1}{2}x - \tfrac{1}{2}y + 1) \cdot a + \ldots && (3.36)\\ &= (-\tfrac{1}{2} \cdot \tfrac{1}{2} - \tfrac{1}{2} \cdot \tfrac{1}{2}\sqrt{3} + 1) \cdot a + \ldots && (3.37)\\ &= (\tfrac{3}{4} - \tfrac{1}{4}\sqrt{3}) \cdot a + \ldots && (3.38) \end{aligned}$$

Using similar derivations as before, we can state that $W \leq \frac{3}{4} - \frac{1}{4}\sqrt{3}$.

□

As before, simulations have been performed for the bitruncated cube. These simulations seem to suggest that $W = \frac{3}{4} - \frac{1}{4}\sqrt{3}$ is tight, since no lower instances have been observed. Together with the original simulation for cubes, the results for bitruncated cubes are shown in Figure 3.7. Whereas W was in the range of 0.120 and 0.142 for cubes, we see that the use of bitruncated cubes cause a massive improvement. For any angle, W has a value between 0.316 and 0.434. This is fairly impressive, since it shows that the preservation quality of such terrains under arbitrary rotation in $\mathbb{R}^3$ is comparable to that of hexagonal grid cells in $\mathbb{R}^2$.

3.5 Open questions

In this chapter we have performed both practical and theoretical measurements to make predictions about the preservation qualities of terrains under rotation. Still, the lack of exact derivations, but mere simulations of W leave the results presented in this chapter inconclusive. Therefore, an open topic for research would be to prove the tightness of W.

Also, using this measurement criterion, one could wonder whether there exist terrain models for $\mathbb{R}^2$ and $\mathbb{R}^3$ that provide better bounds for W than hexagons and bitruncated cubes. This could be accomplished by proving a maximum lower bound for W, preferably as a function of d – the number of dimensions.

So far we have only looked at simple linear interpolation schemes, while it is also possible to perform higher order interpolations.

4 Least-cost paths on grids

In graph theory, the shortest path problem is the problem of finding a path between two vertices in a graph such that the sum of the weight (e.g. the length) of the edges along the path is minimal. Algorithms for calculating shortest paths have found many applications in practice, the most prominent being route planning software and satellite navigation systems.

The shortest path problem cannot be applied to grid-based terrains directly. When looking at the neighbour graph for a grid of hexagonal grid cells, all edges will simply have equal length, meaning that computing shortest paths on these grids is rather uninteresting. In practice many grid-based approaches will use an extended version of the problem called the *least-cost path problem*. In this setting, the algorithm minimises a sum of weights which are attached to the grid cells – the vertices of the neighbour graph. This weight can be interpreted as being the *cost* that is inferred when passing through the grid cell. An example of such a system is Terracost.[8]

The weights attached to the grid cells may be based on individual metrics, or even a combination of them. Consider the case where a railway company is interested in building a new railroad track between two train stations. Not only is it of interest that railroad tracks are built on surfaces that are as flat as possible, the price of land differs from place to place. Computing a path that is good with respect to both these metrics, can be achieved by combining them into a single cost value.

In a more theoretical setting, a least-cost path from point a to point b can be expressed as follows. Assume there is a two-dimensional plane, having a weight function $W : \mathbb{R}^2 \to \mathbb{R}^+ \cup \{0\}$, which maps points on the terrain to their corresponding weight value. A path on a plane can be expressed as a function which maps a distance with respect to its starting point on the path, to a coordinate, thus having type $P : \mathbb{R} \to \mathbb{R}^2$. Because the path function uses the distance as its input, it must progress at a constant speed. This means that for any distance d along the path the following property holds:

$$\lim_{e \to 0} \frac{\|P(d+e) - P(d)\|}{e} = 1 \tag{4.1}$$

The least-cost path involves finding a path function P, which minimizes the following expression:

$$\int_{P^{-1}(a)}^{P^{-1}(b)} W(P(x))\,\mathrm{d}x \tag{4.2}$$

Do observe that for a single terrain, there may be multiple least-cost paths. For example, on a terrain only consisting of grid cells having weight 0, there are an infinite number of least-cost paths. From now on, we shall call least-cost paths having the shortest length the *shortest least-cost paths*.

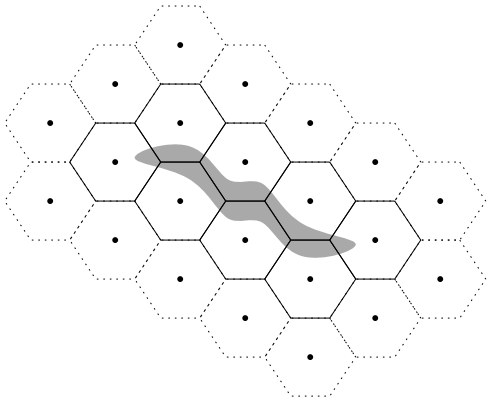


Figure 4.1: A high-cost area that resides between the sample points of the grid, coloured grey. Sample points are shown in black.

Lemma 6 *A shortest least-cost path cannot intersect itself.*

Proof. Assume there exists a shortest least-cost path that intersects itself. Then the length of the path can always be reduced further without increasing the cost of the path by cutting off the part of the path between the intersection point. □

4.1 Properties of least-cost paths on non-interpolated terrains

By the previously given definition, we assume that a terrain is modelled as having an infinite amount of precision. Though in real life terrains usually have a practically infinite precision, terrains using grid layouts cannot provide this. The precision of these terrains is simply bound by their resolution. This imprecision also propagates to the results of the least-cost path calculation. An example of a terrain where this may lead to large imprecision, is shown in Figure 4.1. A least-cost path computed for this terrain may decide to pass straight through the high-cost area, since the high-cost area lies between the sample points. It may be the case that it is less expensive to avoid the high-cost area altogether.

How physical terrains should be converted to sample values on a grid is outside the scope of this thesis, since in practice datasets are obtained through external parties or are constrained by the technique used to sample them. From now on we shall assume that the sampled terrain is exact. We shall interpret the cost value at a centrepoint of a grid cell identical to any point within the grid cell it resides in.

The use of grid terrains where the cost value of a grid cell applies to all points within, leads to the following properties, which we shall refer to later on.

Lemma 7 *A shortest least-cost path on a grid terrain having convex grid cells passes through individual grid cells along the path using only one line segment.*

Proof. Since the grid cells are convex of shape, any two points on the boundary can be connected using a line segment. Because a line segment is the shortest way to connect these two points and the cost value is equal at any point within the grid cell, a shortest least-cost path must pass through the grid cell using only one line segment. □

Do note that the illustrations in this chapter do not use straight lines, as to make them more appealing to the eye.

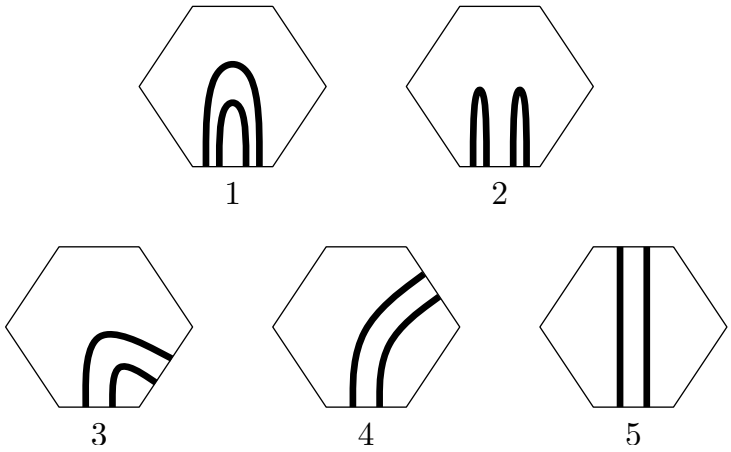


Figure 4.2: The five ways of placing two edges of the path in a hexagon, where both edges share the same pair of edges on the boundary.

Lemma 8 *Consider a grid terrain having convex grid cells, where any two points on the boundary of a grid cell can be connected with a line having at most length l. Then the sum of the length of the edges of a shortest least-cost path that pass through a grid cell cannot exceed l.*

Proof. Consider there is a grid cell, where the sum of the length of the edges that pass through exceeds l. Take the two points on the boundary of the grid cell along the path closest to the beginning and the end of the path. These two points can be connected with an edge of length at most l. This operation cannot increase the weight of the path, while it will reduce its length, thus causing a contradiction. □

Lemma 9 *A shortest least-cost path cannot cross the same boundary edge of a grid cell three or more times consecutively.*

Proof. If there were a shortest least-cost path that crosses the same boundary edge two times in a row, it would imply that one of the grid cells has a lower weight than the other. Three or more times in a row would imply the inequality to hold in both directions, which is a contradiction.

We shall call the pattern of crossing the same boundary edge more than twice in a row 'zigzagging'. □

Lemma 10 *Consider a shortest least-cost path on a grid terrain of hexagonal grid cells. There cannot exist a pair of line segments along this path that intersect the same pair of edges on the boundary of a grid cell.*

Proof. There are five ways in which we can place the two edges of the path in a hexagon, as shown in Figure 4.2. It can be shown that all these five cases cannot exist, as they all conflict with the previously described properties.

For a grid of hexagonal grid cells where the distance between the centrepoints is equal to 1, we know that the largest possible distance between two points on the boundary is $\frac{2}{\sqrt{3}}$. The lengths of the edges in case 4 and 5 are at least $\frac{1}{\sqrt{3}}$ each, meaning it conflicts with Lemma 8. Cases 1 and 3 cannot occur, since the outer edge can simply be rerouted to one of the inner points on the boundary, shortening the length of the path.

Case 2 can develop itself in two different ways, as shown in Figure 4.3. None of these cases can occur in practice. Case 2a cannot occur, since it involves zigzagging. Case 2b would imply the bottom cell has at least two edges of length $\frac{1}{\sqrt{3}}$, contradicting Lemma 8.

□

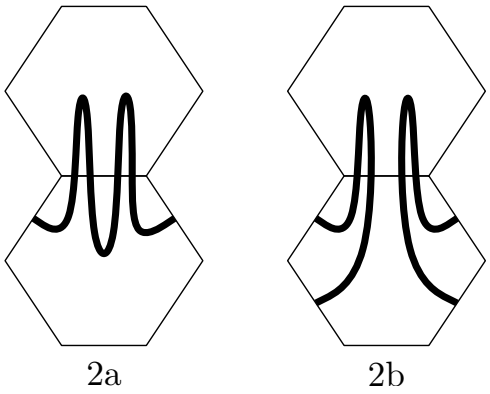


Figure 4.3: Developments of case 2.

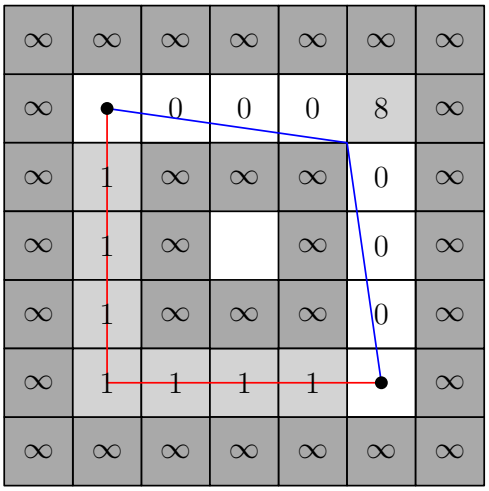


Figure 4.4: An exact least-cost path coloured in blue and an approximated least-cost path coloured in red.

It remains questionable whether these lemmas give a full characterisation of all valid shortest least-cost paths. Still, for a grid terrain of hexagonal grid cells it already places a constant upper bound on the number of times a grid cell is passed through.

4.2 Approximate least-cost path calculation on non-interpolated terrains

In order to provide efficient least-cost path calculation, there are many implementations that do not calculate an exact path across the terrain. Instead of allowing the path to cross through the grid cells arbitrarily, they calculate a path by only allowing straight lines between the centrepoints of the grid cells.

The question is whether such an approximation of least-cost paths yields results that do not deviate much from the exact solution. We can already demonstrate that this is not the case for regular grids where we are only allowed to make cardinal steps. In fact, for this terrain model the approximated least-cost path can have a total cost that is infinitely larger than for an exact least-cost path. This is demonstrated in Figure 4.4, where the exact least-cost path has total cost infinitely close to 0, where it crosses an infinitely small part of the cell having weight 8. The lowest possible total cost for an approximated path is 7.

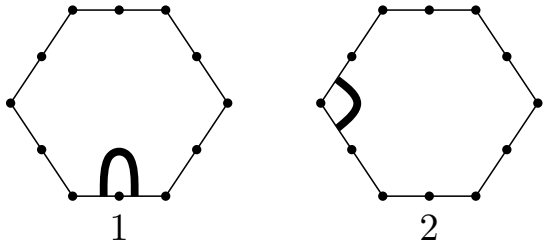


Figure 4.5: Placement of edges of size smaller than $\frac{1}{2\sqrt{3}}$.

4.2.1 Hexagonal grid cells

We shall now demonstrate that terrains using hexagonal grid cells do provide an upper bound on approximated least-cost paths compared to exact paths.

Lemma 11 *Assume there is an exact shortest least-cost path on a terrain of hexagonal grid cells. Let E denote the set of line segments of the shortest least-cost path that pass through each of the grid cells individually. Let $t(c)$ be the sum of the lengths of all the path segments in E that pass through cell c. Also let C be the set of grid cells where $t(c) \geq \frac{1}{2\sqrt{3}}$. The grid cells in C form one connected component.*

Proof. We can prove this lemma by showing that if there is a part of the shortest least-cost path that passes through grid cell c with $t(c) < \frac{1}{2\sqrt{3}}$, then there exists a preceding grid cell p and a succeeding grid cell q along the path that are neighbours (or identical) and $t(p), t(q) \geq \frac{1}{2\sqrt{3}}$.

There are two ways in which paths of at most $\frac{1}{2\sqrt{3}}$ can be placed in a grid cell, as shown in Figure 4.5. This is because $\frac{1}{2\sqrt{3}}$ is equal to half the length of the boundary edges. Of case 1 we can say that since zigzagging is not permitted, the two edges in the neighbouring cell must traverse to other cells. The length of these edges must sum up to at least $\frac{1}{2\sqrt{3}}$, for them to reach the boundary.

Case 2 becomes a bit more complex. If it weren't for the previously given lemmas, the number of possible arrangements of the path would have been infinite. Now if we generate all possible instances in which the path does not cross itself, does not zigzag and there exists no pair of line segments on the path that goes to the same pair of neighbouring grid cells, we obtain the nine cases given in Figure 4.6. Some cases have been omitted, because of horizontal reflection symmetry. Also, to reduce the number of separate cases, the least-cost path can enter and leave this cluster through any edge on the outer boundary of the grid cell – not just the edge used in the picture. Some of these instances are likely impossible to create in practice, but would require additional lemmas to be filtered out.

For all of the nine cases, we can derive that $t(b) \geq \frac{1}{2\sqrt{3}}$. In cases $2d$ to $2i$, the path traverses through cell b before and after passing through c, meaning cell c can safely be removed from the result, without breaking connectivity.

For the first three cases, $2a$ to $2c$, we can prove that $t(a) \geq \frac{1}{2\sqrt{3}}$. Since a and b are adjacent, c can safely be omitted from C without breaking connectivity. □

Theorem 12 *On a terrain consisting of hexagonal grid cells the total cost of the approximated least-cost path between two sample points is at most $2\sqrt{3}$ times the total cost of the exact least-cost path.*

Proof. We can prove this by applying Lemma 11. We can obtain a set of grid cells which is a connected component, where for each grid cell c, $t(c) \geq \frac{1}{2\sqrt{3}}$.

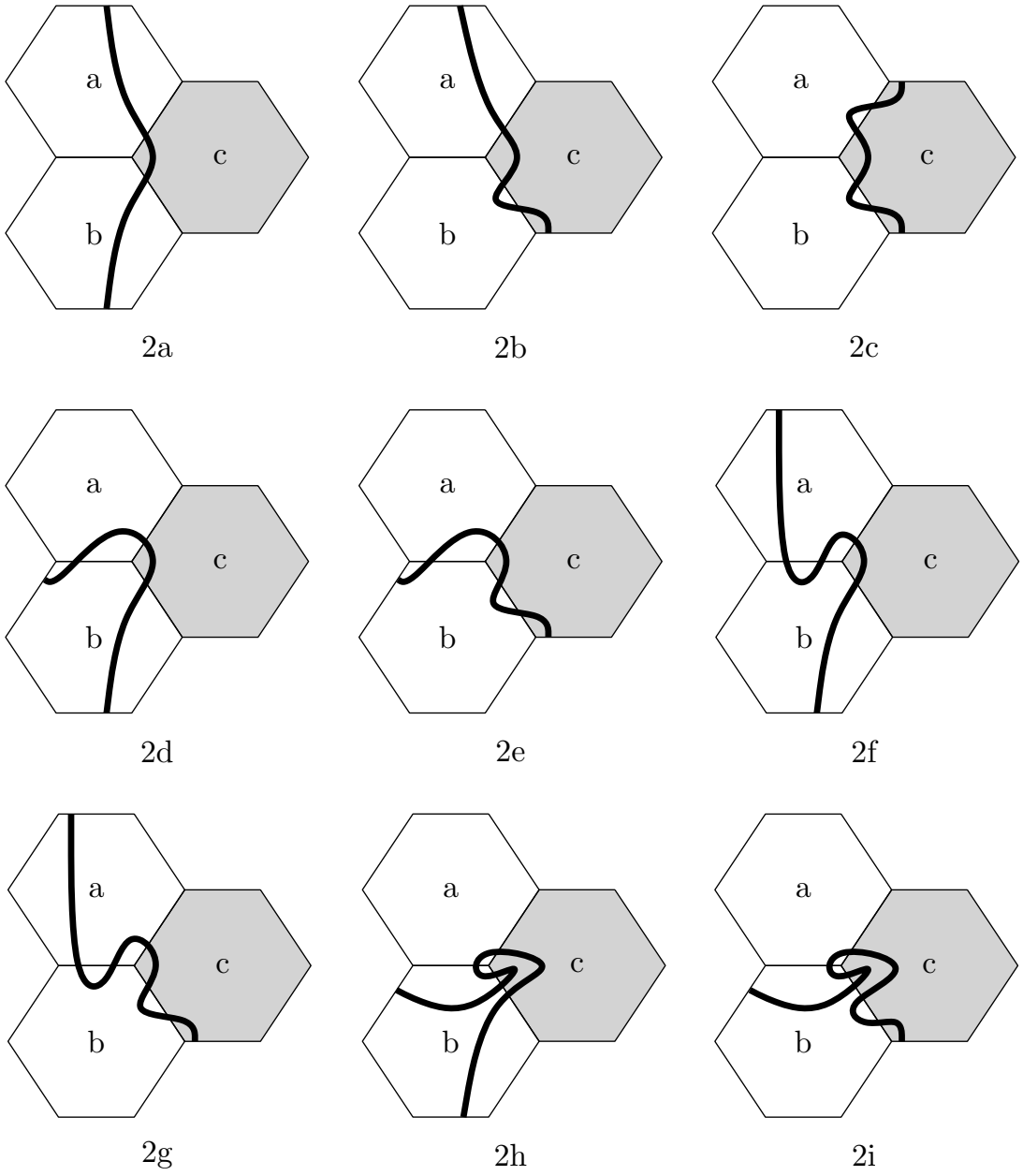


Figure 4.6: Nine cases in which c is traversed in less than $\frac{1}{2\sqrt{3}}$.

Because the set of grid cells is a connected component, it is possible to calculate a path over the grid cells between the two query points. An approximated path can then be generated by connecting the centrepoints of these grid cells.

In the approximated least-cost path, every grid cell of the exact shortest least-cost path is either discarded or traversed with two edges, each having length $\frac{1}{2}$, thus 1 in total. Because all used grid cells were originally traversed with length at least $\frac{1}{2\sqrt{3}}$, the approximated path has at most $2\sqrt{3}$ times its cost. □

4.2.2 Regular grid cells with cardinal and diagonal steps

At the start of this section, we argued that approximated least-cost paths for regular grid cells are not bounded by a constant when compared to the exact least-cost path. For this, we assumed we were only allowed to make steps in cardinal directions. The example we have given in Figure 4.4 does not hold for the case where diagonal steps are also allowed. Just like the terrain model of hexagonal grid cells, it is bounded.

In this section we shall provide a proof for this terrain model that uses the same outline as our proof for hexagonal grid cells. Therefore the proof shall only describe the differences to the original. Let us assume the regular grid cells have size 1×1.

Theorem 13 *On a terrain consisting of regular grid cells the total cost of the approximated least-cost path between two points is at most $2\sqrt{2}$ times the total cost of the exact least-cost path, under the assumption that both rectilinear steps and diagonal steps are allowed.*

Proof. Theorem 12 depends on a set of lemmas. The first two lemmas were independent of the structure of grid cells, meaning only Lemmas 10 and 11 need to be adjusted.

Lemma 10 can trivially be adjusted to prove that a regular grid cell never contains two path segments passing through the same pair of boundary edges, using the same style of case distinction on placements to prove a contradiction. For each of these cases it can be reasoned that a shorter least-cost path must exist.

Lemma 11 can be altered to prove that for every grid cell b where $t(b) < \frac{1}{2}$, there exists a preceding grid cell p and a succeeding grid cell q along the path that are neighbours (or identical) and $t(p), t(q) \geq \frac{1}{2}$. For example, six developments of a grid cell b where an edge passes through a corner of the grid cell, similar to Figure 4.6 are shown in Figure 4.7. By enumerating these six and all other developments, it can be shown that this lemma holds.

We can therefore state that any set of grid cells through which a least-cost path traverses, where all grid cells are removed that are traversed less than $\frac{1}{2}$, is still a connected component. An approximated path can pass through each of the centrepoints of these grid cells, using two path segments of length at most $\frac{1}{\sqrt{2}}$, thus at most $\sqrt{2}$ in total. Therefore the total cost of the path is approximated at most by a factor $\sqrt{2}/\frac{1}{2} = 2\sqrt{2}$. □

4.2.3 Tightness of the worst-case approximation factors

The proofs given previously have placed an upper bound on the approximation factor of approximated least-cost paths for hexagonal grid cells and regular grid cells, under the assumption that diagonal steps are allowed. The question remains whether the obtained bounds of $2\sqrt{3}$ and $2\sqrt{2}$ are in fact tight. Unfortunately, it does not seem possible to create instances that reach these bounds.

For hexagonal grid cells, it only seems to be possible to create instances where the length at which a grid cell is traversed only increases by a factor $\sqrt{3}$, as shown in Figure 4.8(a). For the regular grid cells, we see a similar pattern, where we can only find examples that demonstrate

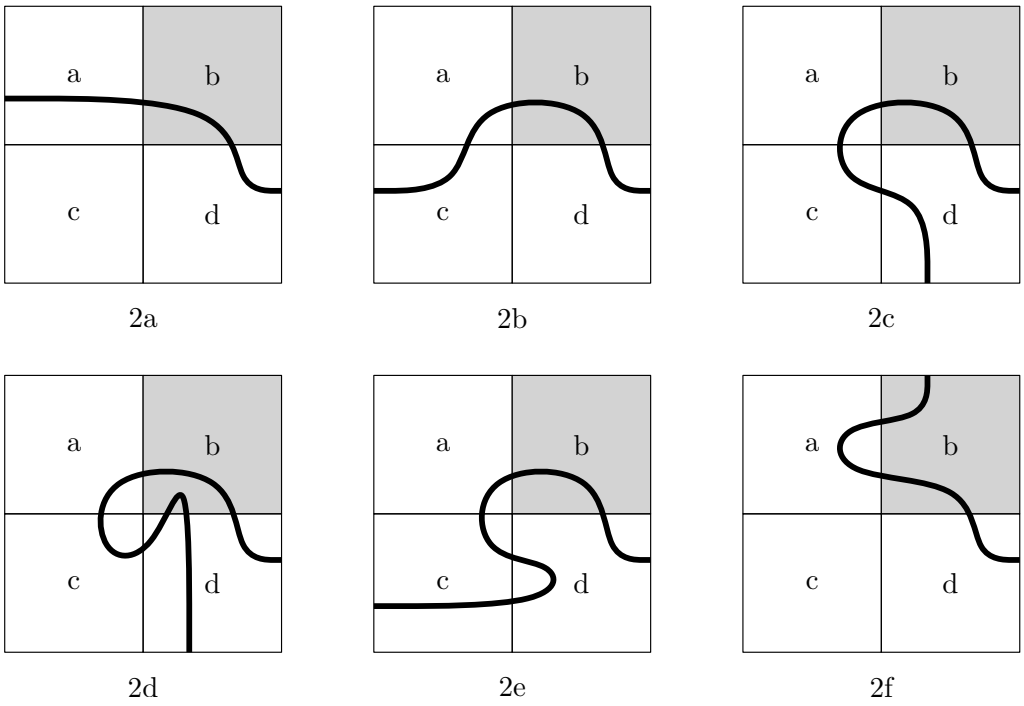
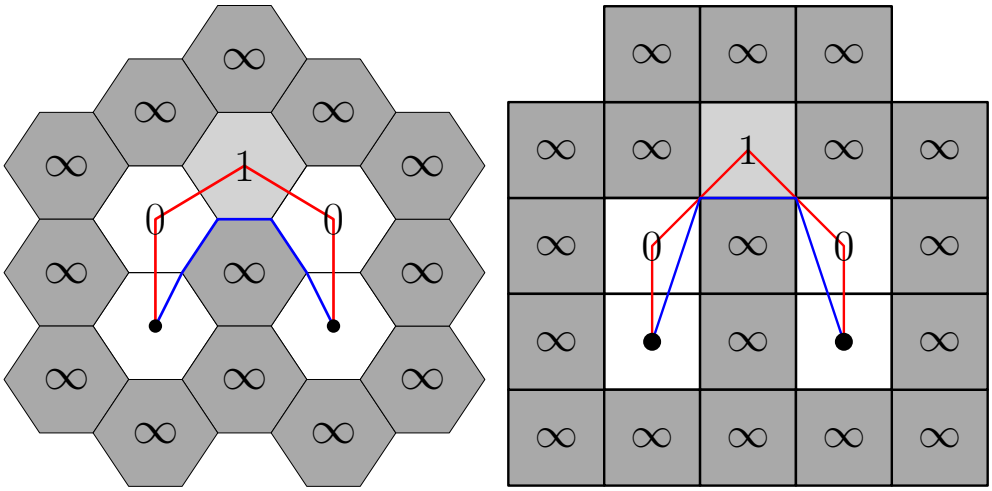


Figure 4.7: Six cases in which b is traversed in less than $\frac{1}{2}$.



(a) Hexagonal grid cells. (b) Regular grid cells (8).

Figure 4.8: Lower bounds for approximation factor for approximated least-cost paths on non-interpolated terrains.

a bound of $\sqrt{2}$, as shown in Figure 4.8(b). As summarised in Table 4.1, we thus have a factor 2 difference between the lower bound and the upper bound.

This discrepancy cannot be eliminated by using the same proof structure. For example, for the regular grid cells with diagonal steps, one would need to prove that the removal of all cells that are traversed with a length less than 1 will not cause the grid cells along the path to become disconnected. A grid having uniform cost as simple as shown in Figure 4.9 already contradicts this premise, as all grid cells are traversed with a length of only $\frac{1}{\sqrt{2}}$ and their omission clearly makes the approximated least-cost path discontinuous.

	Regular (4)	Hexagonal	Regular (8)
Lower bound worst-case approximation factor	∞	$\sqrt{3} \approx 1.73$	$\sqrt{2} \approx 1.41$
Upper bound worst-case approximation factor	∞	$2\sqrt{3} \approx 3.46$	$2\sqrt{2} \approx 2.83$

Table 4.1: Worst-case approximation factors for approximated least-cost paths on non-interpolated terrains.

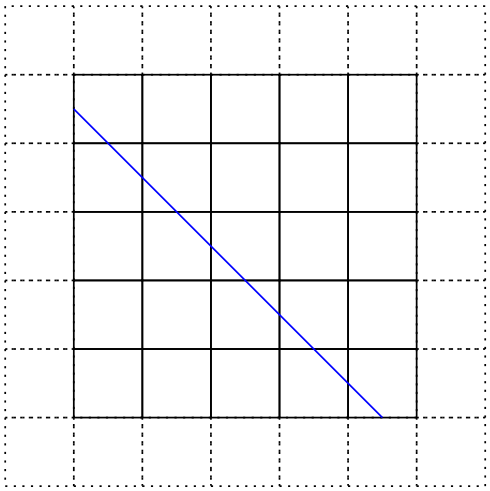


Figure 4.9: A diagonal least-cost path across the grid.

4.3 Approximate least-cost path calculation on interpolated terrains

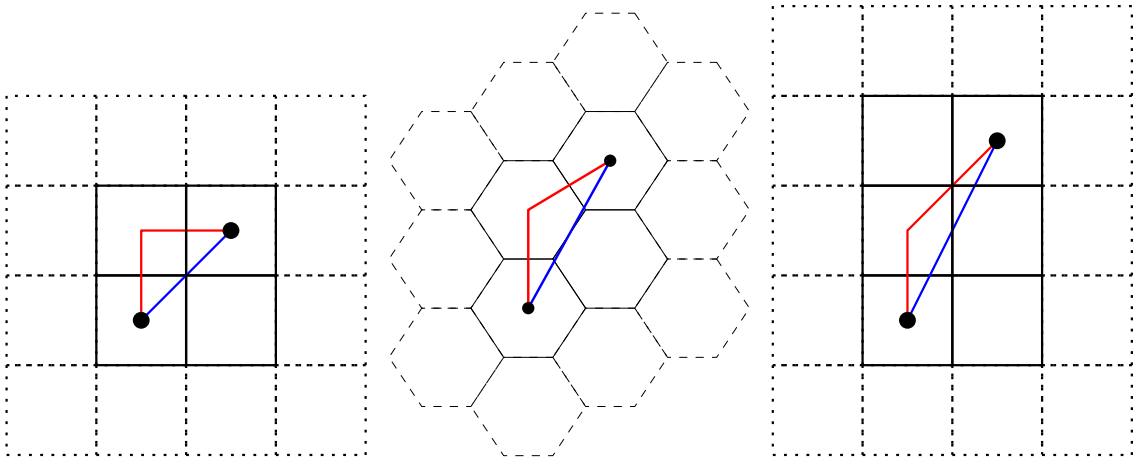
In the previous section we analyzed bounds for the worst-case approximation factor for least-cost paths under the assumption that the weight for an arbitrary point on the terrain corresponds with the value assigned to the centrepoint of the grid cell it resides in. In practice one would rather consider an interpolated – thus smoothed – version of the terrain to be a better representation of the original terrain. In this section we shall assume linear interpolation, as described in sections 3.1 and 3.2.

First of all, we can argue that calculating approximated least-cost paths for interpolated terrains is performed almost identical to non-interpolated terrains. For regular grid cells where only cardinal steps are allowed and hexagonal grid cells, we can easily derive that the added cost of a single step is accounted half to the source grid cell and half to the destination grid cell. For diagonal steps on a regular grid, this is not the case.

Consider the grid shown in Figure 4.8(b). For a non-interpolated terrain, an exact least-cost path algorithm will return a path having total cost 1. If the terrain is considered to be interpolated, there exists no least-cost path having a finite total cost. This is due to the fact that on an approximated least-cost path, the source and destination cells only account for $\frac{1}{3}$ of the added cost, while the intersected neighbours account for $\frac{1}{6}$ each. This can be shown by integrating over the interpolation function V for regular grids diagonally:

$$\begin{aligned}
\int_0^1 V(i,i)\,\mathrm{d}i &= \int_0^1 (1-i)\cdot i\cdot a + i\cdot i\cdot b + (1-i)\cdot(1-i)\cdot c + i\cdot(1-i)\cdot d\,\mathrm{d}i && (4.3)\\
&= \left[\tfrac{1}{2}ai^2 - \tfrac{1}{3}ai^3 + \tfrac{1}{3}bi^3 + ci - ci^2 + \tfrac{1}{3}ci^3 + \tfrac{1}{2}di^2 - \tfrac{1}{3}di^3\right]_{i=0}^{1} && (4.4)\\
&= \tfrac{1}{6}a + \tfrac{1}{3}b + \tfrac{1}{3}c + \tfrac{1}{6}d && (4.5)
\end{aligned}$$

It is unknown whether proofs similar to the ones from the previous section can be adjusted to provide bounds on a worst-case approximation factor for least-cost paths on interpolated

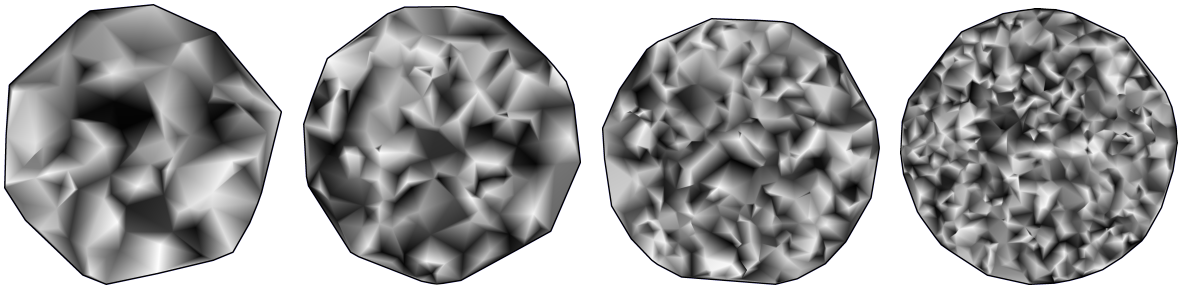


(a) Regular grid cells (4). (b) Hexagonal grid cells. (c) Regular grid cells (8).

Figure 4.10: Lower bounds for approximation factor for approximated least-cost paths on interpolated terrains.

	Regular (4)	**Hexagonal**	**Regular (8)**
Lower bound worst-case approximation factor	$\sqrt{2} \approx 1.41$	$\frac{2}{\sqrt{3}} \approx 1.15$	$\frac{1+\sqrt{2}}{\sqrt{5}} \approx 1.08$
Upper bound worst-case approximation factor	∞	∞	∞

Table 4.2: Worst-case approximation factors for approximated least-cost paths on interpolated terrains.



(a) 125 vertices (b) 250 vertices (c) 500 vertices (d) 1000 vertices

Figure 4.11: Random TINs.

terrains. We can however provide weak lower bounds on the approximation factor by looking at the case where the exact least-cost path traverses between two angles on a terrain having uniform cost, as shown in Figure 4.10. These cases lead to the lower bounds given in Table 4.2.

4.4 Measuring terrain model quality using random TINs

In addition to the theoretical analysis performed previously, we shall now look at measurements of least-cost paths performed on TINs, which are generated by calculating the Delaunay triangulation of a randomly generated pointset. Each vertex on the TIN has a uniformly distributed random cost value between 0 and 1. Each of the TINs has a circular shape, but have a varying number of vertices. The four TINs generated are shown in Figure 4.11.

In order to perform a fair comparison, one should compare the approximated least-cost paths with an exact least-cost path on the same dataset, but unfortunately such a compari-

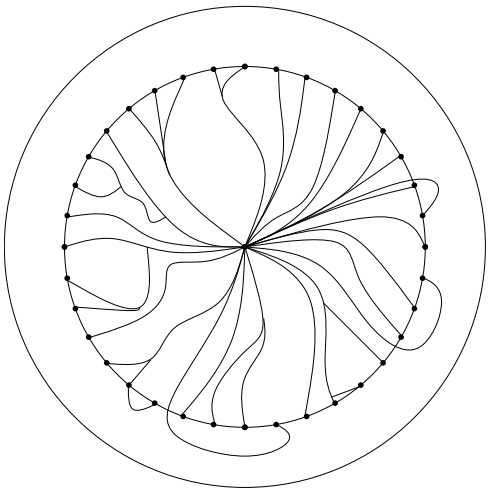


Figure 4.12: Least-cost paths from centrepoint to circle of measurement points.

TIN size	Regular (4)	Hexagonal	Regular (8)	Regular (8) + I
125 vertices	0.041041	0.016837	0.008697	0.008692
250 vertices	0.037515	0.015284	0.007236	0.007238
500 vertices	0.027110	0.011140	0.005391	0.005389
1000 vertices	0.028249	0.009577	0.005436	0.005448

Table 4.3: Coefficients of variation observed for least-cost path calculations for various randomly generated TINs.

son is hard to achieve, as calculating least-cost paths on TINs is far from trivial.[10] Therefore we use a different method of comparing our three terrain models.

For each of the datasets, we calculate the least-cost paths from the centrepoint of the grid to 360 equidistant measurement points that are placed on a circle which has a radius $\frac{3}{4}$ the size of the TIN itself, as depicted in Figure 4.12. Because the least-cost path algorithms work on grids – not TINs – we repeat the same test for 360 samplings of the TIN under different angles. This means that the least-cost path to each of the measurement points is calculated under 360 different angles. Each grid is sampled from the TIN at a resolution of 2^{24} grid cells.

For each of the measurement points, we calculate the coefficient of variation ($c_v = \frac{\sigma}{\mu}$) of the 360 least-cost paths that have been calculated. For exact least-cost paths, this coefficient should of course be 0, but the lack of resolution and angles in which an approximated least-cost path can progress, will yield different results for each angle.

For each terrain model, the average coefficient of variation for the 360 measurement points is provided in Table 4.3. As we can see, the best results are obtained when using a regular grid, but only if diagonal steps are allowed. The coefficient of variation for hexagonal grid cells is approximately twice as large, but still 2.5 times smaller than regular grid cells where only cardinal steps are allowed. The last column of the table uses the alternative interpolation method for diagonal steps as described in the previous section. As we can see, it has little to no influence on the final results.

The approximation algorithms are implemented in such a way that they never underapproximate the total cost of a least-cost path. Therefore we can use the results acquired by these tests to determine whether the theoretical bounds can in fact be reached in practice. In Table 4.4 we provide the highest observed ratio between the shortest least-cost path and the longest least-cost path computed for a single measurement point. Though we see the same trend as in Table 4.3, we can also observe that these ratios come pretty close to the lower bounds we

TIN size	Regular (4)	Hexagonal	Regular (8)	Regular (8) + I
125 vertices	1.27433	1.12197	1.05711	1.05713
250 vertices	1.30345	1.09792	1.05774	1.05776
500 vertices	1.26135	1.07666	1.03520	1.03523
1000 vertices	1.22048	1.07947	1.03064	1.03064

Table 4.4: Ratio between shortest and longest path observed for a single measurement point.

saw for interpolated terrains. Because the sampled grids are based on flat surfaces of triangles and would thus be an accurate representation of an interpolated terrain, this is an interesting phenomenon. It doesn't rule out that the lower bounds for linearly interpolated terrains are in fact tight.

4.5 Open questions

In this chapter we have seen some fundamental analysis of least-cost paths for various types of two-dimensional terrain models. Even though we have already demonstrated that for some terrain models approximated least-cost paths are bounded by a constant approximation factor, we have not managed to prove the tightness of this bound. Moreover, no upper bounds have been determined for interpolated terrains, though benchmarks haven't managed to produce instances worse than the derived lower bounds.

In this chapter we have only restricted analysis to two-dimensional terrain models. One might be interested in performing similar analysis for terrain models in $\mathbb{R}^3$.

5 Flow accumulation on grids

In addition to the applications of grid terrains discussed previously, we shall now consider the calculation of flow accumulation. Flow accumulation typically answers the following question. Given a grid terrain T, where each grid cell is subject to a certain amount of precipitation $P(c)$, how much of this precipitation will flow through a specific grid cell before reaching its final destination (e.g. a valley, river or lake)?

Essentially, this problem is solved by translating a dataset (e.g. an elevation map) to a directed graph that indicates whether precipitation may flow out of a grid cell, and if so into which of its neighbours. In the general case, water always tends to traverse downward, so it is safe to assume that this graph is acyclic; a DAG.

In this chapter we shall look at how different terrain models and techniques affect the quality of flow accumulation. First we shall only consider the case where each cell can only propagate precipitation to one of its neighbours as shown in Figure 5.1, called *single-flow-direction* (SFD). Later on, we shall also discuss *multiple-flow-direction* (MFD).

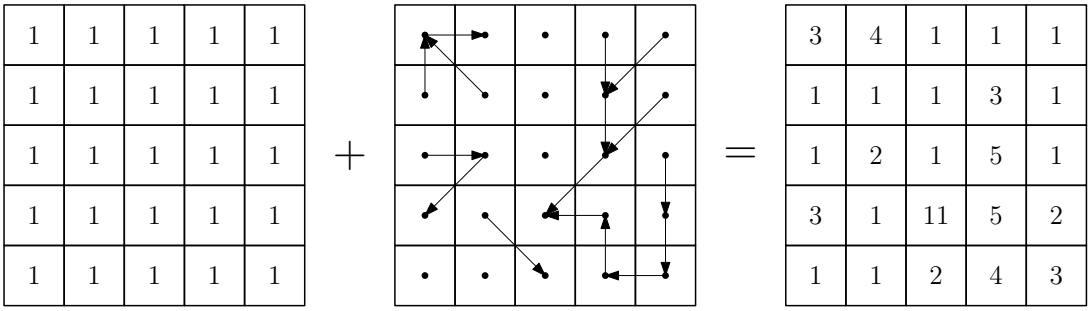


Figure 5.1: An example of a computation of flow accumulation for SFD.

Though algorithms for calculating flow accumulation are given in this chapter, they are merely provided for completeness. I/O efficient versions of these algorithms can be found in an article written by Haverkort and Janssen[7], but also in the TerraFlow suite.[17]

5.1 Calculating flow accumulation for single-flow-direction

Algorithms that calculate the flow accumulation for every cell on a grid, are trivial to write. As the flow on the terrain can be expressed as a DAG, we can use topological sorting algorithms to process the input in such a way that precipitation can be propagated across the entire terrain in a single pass.

Still, even without topological sorting it is possible to calculate the flow accumulation, as demonstrated in Algorithms 3 and 4. The algorithm works by processing each of the cells individually. For each of these cells we propagate the flow accumulation to its out-neighbour, if and only if the flow accumulation for the cell itself is fully computed. This algorithm is identical to the naïve algorithm given in Janssen's paper, except that it is decomposed in two separate routines. This decomposition shall later be used to support MFD.

Algorithm 3 NAÏVEACCUMULATION

```
1: for each cell c ∈ T do
2:         FlowAcc[c] ← P(c)
3:         mark[c] ← false
4: for each cell c ∈ T do
5:         if mark[c] = false then
6:                 MARKCELL(c)
```

Algorithm 4 MARKCELL(Cell c)

```
1: if there exists an in-neighbour of c called d, where mark[d] = false then
2:         {The value of c has not yet been determined.}
3:         return
4: mark[c] ← true
5: if cell c has an out-neighbour, called d then
6:         FlowAcc[d] ← FlowAcc[d] + FlowAcc[c]
7:         MARKCELL(d)
```

5.1.1 Comparison of various terrain models

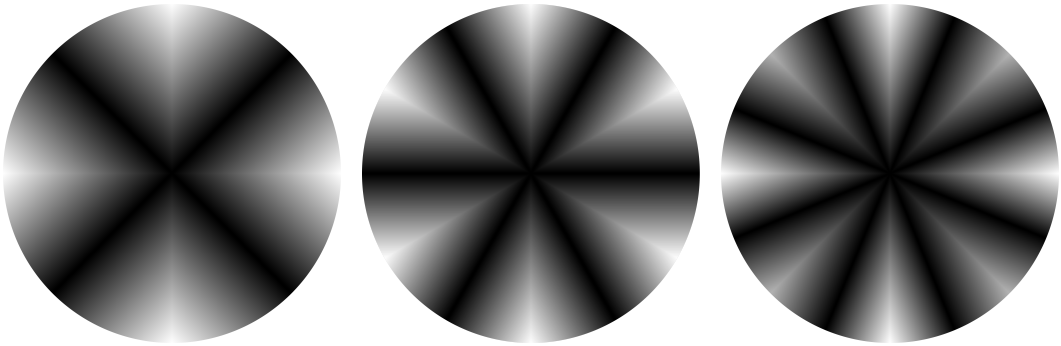
As we will be applying the previously described algorithm on different terrain models, we must construct a dataset that is not biased towards a specific terrain model. Also, the dataset must not depend on in-flow from the boundary of the terrain. For example, a regular terrain where all the precipitation flows to the centrepoint may already be unfair for regular grid cells, depending on the angle at which the grid is placed, since diagonal steps cover a larger distance than axis-aligned steps.

The easiest and probably most model-independent terrain one could think of, would be a terrain consisting of a perfectly circular hill on which precipitation is released uniformly. Theoretically, each point on a circle centered at the top of the hill must have an equal amount of flow accumulation. Obviously, the angle at which precipitation flows at a certain point is identical to the angle the point has respective to the top of the hill. In this comparison we shall round this angle to the nearest neighbour.

In this section we will be comparing three terrain models:

- A terrain consisting of regular grid cells, where precipitation can flow cardinally, thus in four directions,
- A terrain consisting of hexagonal grid cells, where precipitation can flow in six directions,
- A terrain consisting of regular grid cells, where in addition to the first terrain model, precipitation can also flow diagonally, thus in eight directions.

By applying the previously described algorithms, we obtain the results shown in Figure 5.2. As we can see, these results barely resemble the uniform distribution of flow we anticipated.



(a) Regular grid cells, cardinal flow.

(b) Hexagonal grid cells, flow in six directions.

(c) Regular grid cells, flow in eight directions.

Figure 5.2: Flow accumulation on a round hilltop. Black indicates a low amount of accumulation, while white regions correspond with a large amount of accumulation.

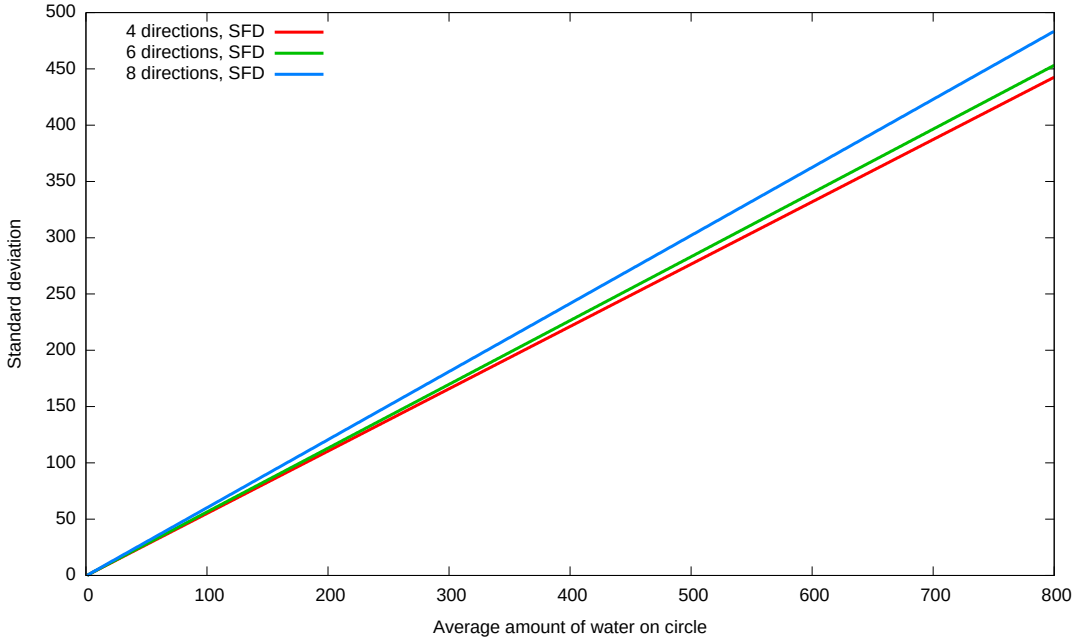


Figure 5.3: Standard deviation of flow accumulation on the boundary of a circle.

As we round the angle of flow to the nearest neighbour, we effectively turn our circular hill into a pyramid, where the centre of each face collects more precipitation than the corners.

Now that we have calculated the flow accumulation for each terrain model, we can calculate the mean value and standard deviation observed on a circle having a variable radius. Because the density of grid cells of the terrain models is incomparable, we disregard the actual radius, but compare the standard deviation by mean value. This way we obtain the graph shown in Figure 5.3.

This graph raises some interesting observations. First of all, when using regular grid cells where water can flow in eight directions, the standard deviation is the largest. This is due to the fact that flow in the four diagonals goes faster than for the rectilinear sides. Contrary to what we hoped, we can also observe that the hexagonal grid model performs worse than the regular grid model.

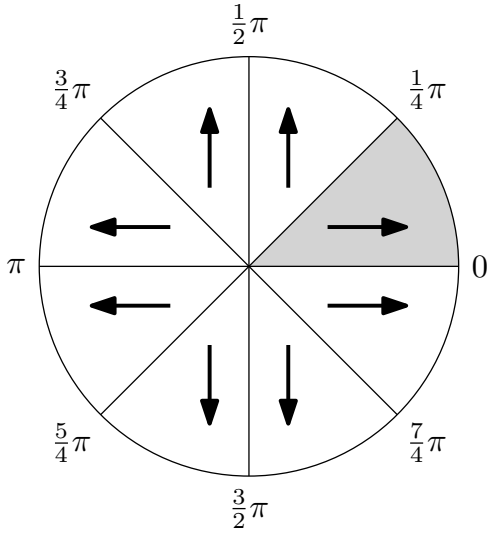


Figure 5.4: Eight segments of waterflow in four directions.

5.1.2 Derivation of coefficient of variation

In the previous section, we have seen that using the naïve flow accumulation algorithm, simple regular grid cells where flow can only be performed in four directions offers the best results for our benchmark. Also, we saw that the standard deviation scaled linear with respect to the mean value of the sample values on the boundary. We shall now prove these properties formally, by deriving exact equations for the mean (μ), standard deviation (σ) and coefficient of variation ($c_v = \frac{\sigma}{\mu}$).

In this section we shall not derive this constant for flow accumulation in eight directions. Of course, it can be derived using a similar approach. For this model, care must be taken to distinguish flow in rectilinear and diagonal directions, as diagonal steps speed up the flow by a factor $\sqrt{2}$.

Theorem 14 *For a terrain having regular grid cells where water can only flow in cardinal directions, the coefficient of variation (c_v) of flow accumulation for a certain circle of radius R placed around a circular mountaintop is* $\frac{\sqrt{-48+32\sqrt{2}-2\pi+\pi^2}}{4(\sqrt{2}-1)} \approx 0.55358$.

Proof. As we can see in Figure 5.4, our mountaintop consists of eight segments that are all symmetric. Therefore, it is sufficient to derive the coefficient of variation for only one of the eight segments. In this proof we shall consider the segment indicated in grey.

Let us define a continuous function A that for an angle on the boundary calculates the amount of accumulated water. This amount is equal to the horizontal distance between the boundary and the diagonal edge. This property can simply be expressed by the following equation:

$$A(\varphi) \quad = \quad (\cos(\varphi) - \sin(\varphi))R \tag{5.1}$$

Now we can derive the mean amount of flow accumulation on the boundary and its standard deviation:

$$\begin{aligned}
\mu &= \tfrac{4}{\pi} \cdot \int_0^{\frac{\pi}{4}} A(\varphi)\,\mathrm{d}\varphi && (5.2)\\
&= \tfrac{4}{\pi} \cdot (\sqrt{2}-1)R && (5.3)\\
\sigma &= \sqrt{\tfrac{4}{\pi} \cdot \int_0^{\frac{\pi}{4}} (A(\varphi)-\mu)^2\,\mathrm{d}\varphi} && (5.4)\\
&= \sqrt{\tfrac{-48+32\sqrt{2}-2\pi+\pi^2}{\pi^2} R^2} && (5.5)\\
&= \tfrac{\sqrt{-48+32\sqrt{2}-2\pi+\pi^2}}{\pi} R && (5.6)
\end{aligned}$$

Combining these results, we obtain the following coefficient of variation:

$$\begin{aligned}
c_v = \tfrac{\sigma}{\mu} &= \tfrac{\sqrt{-48+32\sqrt{2}-2\pi+\pi^2}}{4(\sqrt{2}-1)} && (5.7)\\
&\approx 0.55358 && (5.8)
\end{aligned}$$

□

Theorem 15 *For a terrain having hexagonal grid cells, the coefficient of variation (c_v) of flow accumulation for a certain circle of radius R placed around a circular mountaintop is approximately* 0.56679.

Proof. Reusing the same structure as the previous theorem, we can also calculate the coefficient of variation for a terrain where water flows in six directions, thus twelve segments. Again, we start by defining function A. As the diagonal edge of our segment is now slightly steeper, the definition changes accordingly.

$$A(\varphi) = (\cos(\varphi) - \sqrt{3}\sin(\varphi))R \quad (5.9)$$

As before, we can now use this function to calculate the mean and the standard deviation of the amount of accumulated water on the boundary. Even though an exact derivation of the standard deviation is possible, it is omitted, because of its vast complexity.

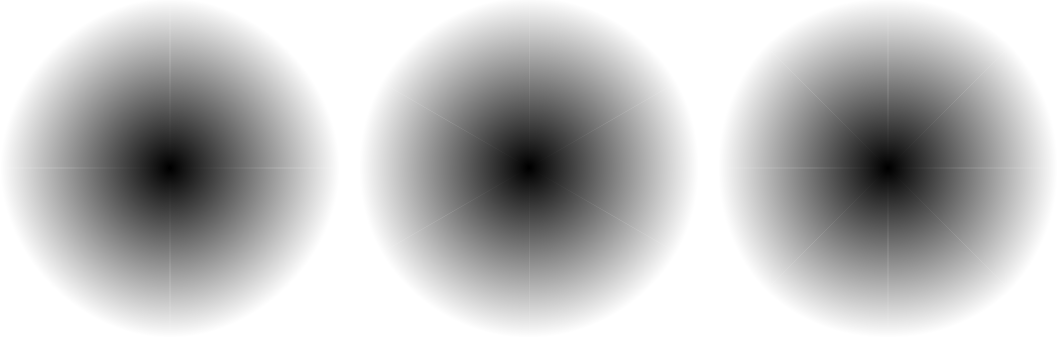
$$\begin{aligned}
\mu &= \tfrac{6}{\pi} \cdot \int_0^{\frac{\pi}{6}} A(\varphi)\,\mathrm{d}\varphi && (5.10)\\
&= \tfrac{6}{\pi} \cdot (2-\sqrt{3})R && (5.11)\\
\sigma &= \sqrt{\tfrac{6}{\pi} \cdot \int_0^{\frac{\pi}{6}} (A(\varphi)-\mu)^2\,\mathrm{d}\varphi} && (5.12)\\
&\approx 0.29005R && (5.13)
\end{aligned}$$

This leads to the following coefficient of variation:

$$\begin{aligned}
c_v = \tfrac{\sigma}{\mu} &\approx \tfrac{0.29005R}{\frac{6}{\pi}\cdot(2-\sqrt{3})R} && (5.14)\\
&\approx 0.56679 && (5.15)
\end{aligned}$$

□

As mentioned before, these coefficients are independent of R – thus constant, which explains the linearity of Figure 5.3. In fact, the constants given in these two theorems can almost exactly be obtained by using the results from this benchmark.



(a) Regular grid cells, cardinal flow.

(b) Hexagonal grid cells, flow in six directions.

(c) Regular grid cells, flow in eight directions.

Figure 5.5: Flow accumulation on a round hilltop, modeling the waterflow using a DAG.

5.2 From forests to DAGs: multiple-flow-direction

So far we have only compared terrain models by considering SFD. This model behaved badly. In fact, it behaved so bad that the coefficient of variation exceeds 0.5 for all terrain models analysed. Therefore we shall repeat the same tests for MFD.

When using SFD, the flow between grid cells can be expressed as a forest. A cell can have multiple in-neighbours, but only a single out-neighbour. For MFD, a cell can have multiple out-neighbours, meaning the flow relationship is expressed using a full DAG. To account for this, the provided algorithm must be changed accordingly. Algorithm 5 is a modified version of the MARKCELL function, altered to propagate its $FlowAcc$ value to multiple neighbours according to an edge weight function W.

Algorithm 5 MARKCELLDAG(Cell c)

1: **if** there exists an in-neighbour of c called d, where $mark[d] =$ **false then**
2: {The value of c has not yet been determined.}
3: **return**
4: $mark[c] \leftarrow$ **true**
5: **for** each cell d, being an out-neighbour of c **do**
6: $FlowAcc[d] \leftarrow FlowAcc[d] + W(cd) \cdot FlowAcc[c]$
7: **for** each cell d, being an out-neighbour of c **do**
8: **if** $mark[c] =$ **false then**
9: MARKCELLDAG(d)

Now we have to decide how to choose our weight function. For our specific testcase – the perfect round hill – the simplest way would be to spread the flow between two adjacent neighbouring grid cells, according to a ratio based on the actual angle. Using this procedure we obtain the results shown in Figure 5.5.

These renderings are close to what we expect. To the eye they seem like a uniform accumulation of precipitation in any direction. Though hardly visible on paper, each of these maps contain a set of hairlines that pass through the origin, where the amount of accumulation is very large compared to other sample points. This effect is depicted in Figure 5.6. Precipitation that flows across the terrain along one of the axes will pass to a single neighbouring cell, while all other precipitation will be scattered. For the rest of this section we shall disregard these cases and omit them from any measurements.

Similar to the measurements performed for SFD, it has been observed that the mean amount

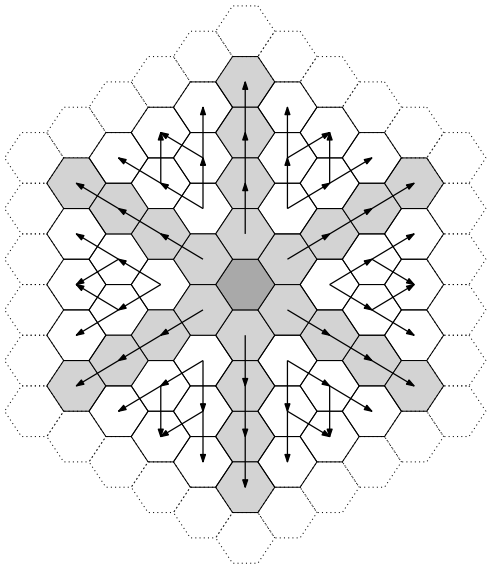


Figure 5.6: Hairlines induced by dividing flow to two neighbours. The dark grey cell corresponds with the top of the hill.

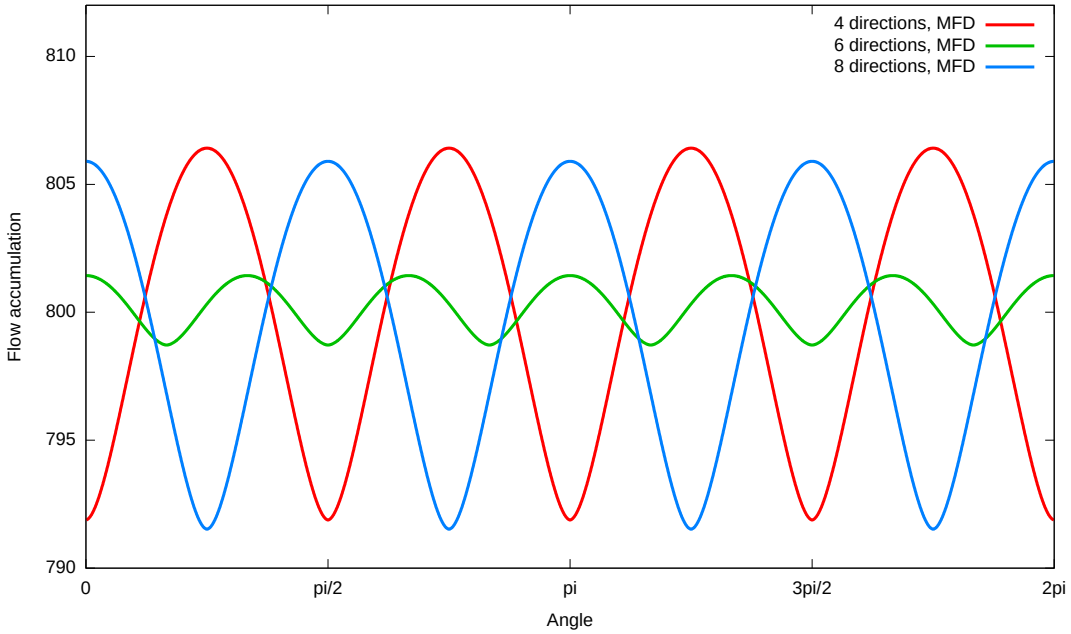


Figure 5.7: Accumulated flow as a function of the angle, for a fixed mean value of approximately 800.

of accumulated flow on a circle and its standard deviation scale linearly with respect to the radius of a circle centered around the hilltop. As can be seen in Figure 5.7, the amount of accumulated water is pretty close to uniform, though a terrain having hexagonal grid cells performs better than one having regular grid cells. For an approximate mean value of 800, they have $c_v = 0.00116$ and $c_v = 0.00616$, respectively.

5.2.1 Constructing gradient-based weight functions

So far we have only performed measurements for weight functions that were based on the angle of steepest descent. Typically, flow accumulation algorithms are applied to existing elevation maps. For these maps, flow direction is initially unknown. Therefore we shall investigate whether it is possible to recompute this angle of steepest descent based on an elevation map.

From a mathematical point of view, we are interested in computing the *gradient* for a given point on the grid. The gradient is a vector field that points in the direction of the greatest rate of increase in elevation. By taking the inverse of this vector field, we thus obtain the direction of greatest rate of decrease. The gradient is defined as follows:

$$\nabla f(x,y) = \begin{bmatrix} \frac{\partial}{\partial x} f(x,y) \\ \frac{\partial}{\partial y} f(x,y) \end{bmatrix} \tag{5.16}$$

Now the problem is that our terrain is not defined as a function that we can use to calculate the gradient, but as a set of sample points. Still, we can work around this by creating a polynomial function f that approximates the shape of the terrain given. For example, for a regular grid it is possible to create a polynomial function that is equal in value for point $\begin{bmatrix} x & y \end{bmatrix}$ and all its neighbours, having the following shape:

$$f(x,y) = A + Bx + Cx^2 + Dy + Exy + Fx^2y + Gy^2 + Hxy^2 + Ix^2y^2 \tag{5.17}$$

Now assume we want to develop this function centered around the sample point at position $\begin{bmatrix} 0 & 0 \end{bmatrix}$. By solving A to I as a set of linear equations, we obtain their valuations in terms of the sample values of the grid cell and its neighbours. The gradient then becomes:

$$\begin{aligned} \nabla f(0,0) &= \begin{bmatrix} B \\ D \end{bmatrix} && (5.18) \\ &= \begin{bmatrix} \frac{1}{2}(f(1,0) - f(-1,0)) \\ \frac{1}{2}(f(0,1) - f(0,-1)) \end{bmatrix} && (5.19) \end{aligned}$$

In other words, the gradient can simply be obtained by calculating the difference between the neighbouring grid cells on the horizontal and vertical axis.

Unfortunately it is a lot harder to perform a similar derivation for a grid of hexagonal grid cells. A hexagonal grid cell and its six neighbours are placed at 3 distinct x-coordinates and 5 distinct y-coordinates. This means that a polynomial function of at most 15 terms is sufficient to cause an identical valuation of the seven sample points. The problem with this approach is that if one would attempt to solve a set of 7 linear equations of 15 terms, the resulting system will consist of 8 free variables. This makes it hard in practice to make a fair comparison.

Even worse, the use of gradient-based weight functions is very likely to break one of the very first assumptions we made about input, namely that the flow relationship can be expressed as a DAG. The use of gradient-based weight functions can easily cause cyclic flow, as shown in Figure 5.8. Here we model a very simple one-dimensional terrain model. For both sample points 1 and 2, the shape of the terrain can be approximated using the polynomial function $f(x) = \frac{1}{2}x^2 - \frac{3}{2}x + 3$. As we can see, a gradient-based weight function will simply cause precipitation on sample point 1 to flow to sample point 2, while sample point 2 will send this back to sample point 1.

Fortunately our hilltop benchmark doesn't feature any valleys, meaning gradient-based weight functions can still be applied to this dataset. Simulations for regular grids have shown that for this dataset the coefficient of variation (c_v) is indistinguishable to the values presented for our angle-based weight function.

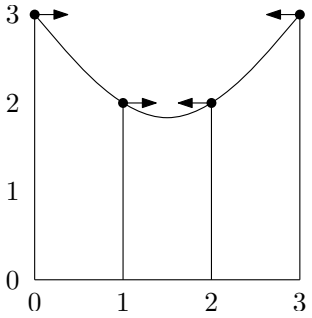


Figure 5.8: Simple one-dimensional terrain model, where water flows cyclic.

5.2.2 Measuring existing slope-based weight functions

Now that we have discussed the practical implications of using gradient-based weight functions, we shall present a method for calculating flow accumulation for arbitrary elevation maps, in such a way that the flow relationship is guaranteed to be acyclic. One of the most common ways to accomplish this, is by using slope-based weight functions. Slope-based weight functions are designed in such a way that water is only allowed to flow to neighbouring grid cells that have a lower elevation, meaning that cycles cannot occur.

In this section we shall consider the work done by Quinn et al.[14], Freeman et al.[4] and Qin et al.[13]. All previously referenced literature base their work on a slope-based weight function, that calculates the weight for neighbour cell i as follows:

$$W_i = \frac{(S_i)^p \times L_i}{\sum_{n \in N} (S_n)^p \times L_n} \tag{5.20}$$

In this expression, the N set denotes all neighbours of a grid cell. S corresponds with the negative slope of a neighbouring cell (e.g. 0 for a flat surface, 1 for a slope of $-45°$). If the neighbouring cell has a positive slope – thus having a higher altitude – it has a slope of 0. L denotes a weight factor to be attached to a specific neighbouring grid cell. The work of Quinn and Qin suggests this weight factor to be set to $\frac{1}{\sqrt{2}}$ for diagonal neighbours and 1 otherwise, as diagonal steps cover a larger distance. Freeman proposes a uniform assignment. In this section we shall use the former definition, as the latter is discussed later on.

The most important difference between the slope-based weight functions, is how the exponent p is assigned:

- Quinn's weight function uses no exponent, hence $p = 1$.
- Freeman observed that $p = 1.1$ offers the most uniform dispersion.
- Qin et al. propose an exponent that is a function of the steepest descent. They perform measurements using $p = 1.1 + 8.9 \times \min(e, 1)$, where e is the maximum slope of all of the neighbouring grid cells.

From now on, we shall refer to these approaches as MFD-Quinn, MFD-Freeman and MFD-md, respectively. It must be noted that an assignment of $p = \infty$ would yield identical results to SFD, as the neighbour with the largest negative slope will always be preferred exclusively.

We can now perform a similar benchmark as we have done previously – by simulating a perfectly round hilltop and calculating the average coefficient of variation observed for circles centered at the hilltop. As the slope-based weight function depends on elevation levels to calculate the slope between two adjacent grid cells, we define the shape of our mountain as a cone, where the difference in elevation compared to the hilltop is equal to the Euclidean

Terrain model	Regular (4)	Hexagonal	Regular (8)
MFD-Quinn	0.09780	0.04199	0.02845
MFD-Freeman	0.05901	0.01689	0.01692
MFD-md	0.50616	0.41533	0.23057

Table 5.1: Coefficients of variation observed for different slope-based weight functions for MFD.

	p	c_v
Regular (4)	1.2252	0.03340
Hexagonal	1.1469	0.01244
Regular (8)	1.0984	0.00912

Table 5.2: Measured optimal values for p.

distance from the top. By applying the flow accumulation algorithm, we obtain the results provided in Table 5.1. Renderings of the terrains are given in Figure 5.9.

Even though the renderings of MFD-Quinn and MFD-Freeman come close to a rendering of a round hilltop, the MFD-md slope-based weight function seems not to perform the way we expect it would. The flow accumulation resembles more the shape of a flower than a round hilltop. This is due to the fact that the hill we have simulated is rather steep, meaning p often reaches values close to the upper bound of 10. This exposes an interesting property of MFD-md. If the elevation data of the entire terrain is multiplied by a constant factor, the flow accumulation algorithm will yield different results. Therefore it is likely that this approach requires tuning for individual terrain datasets.

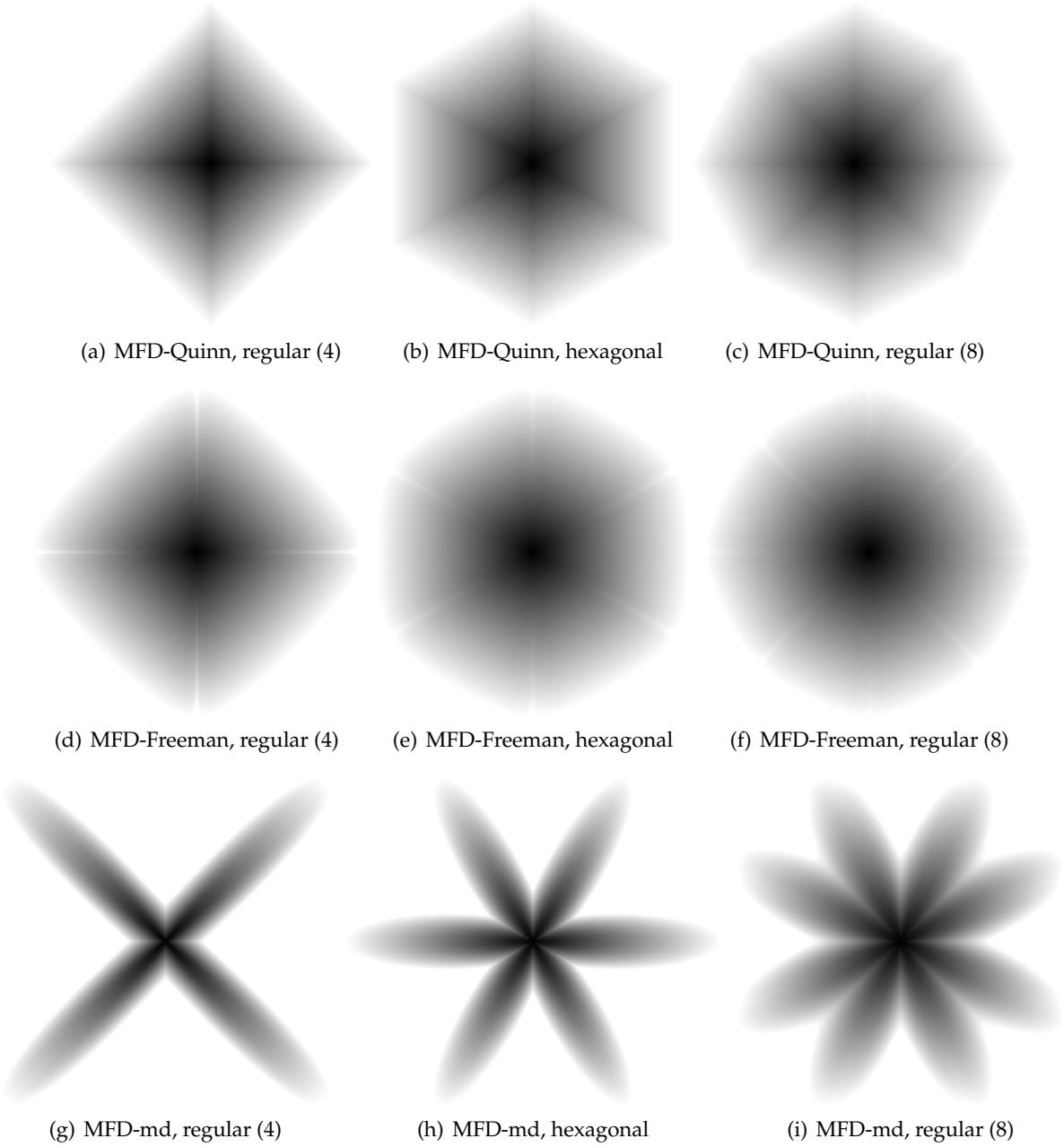
Disregarding the results from MFD-md and limiting our research to constant values of p, we can see that the exponent p can be used to influence the *roundness* of our renderings. For the hexagonal terrain model and the regular grid model where we do allow diagonal flow, we see that Freeman's observation that $p = 1.1$ provides good results can be reproduced in this benchmark. The results show that the standard deviation is less than 1.7% of the average value of all points on a circle centered at the hilltop.

5.2.3 Optimising the weight function's exponent

Now that we have measured the coefficients of variation for existing slope-based weight functions, we may ask ourselves whether there are assignments for p that yield even lower standard deviations. After performing automated tests, we can conclude the following:

- Freeman's approach of assigning uniform values to L seems to yield better results for our hilltop benchmark.
- Freeman's assignment of $p = 1.1$ seems close to the optimal value measured for the regular terrain model, with diagonal steps, which is $p = 1.0984$.
- The assignment of p is dependent of the terrain model used, as different optimal values have been derived for the other terrain models.

The optimal values of p measured, together with their coefficients of variation are provided in Table 5.2. Renderings of the hilltop using these constants are shown in Figure 5.10.

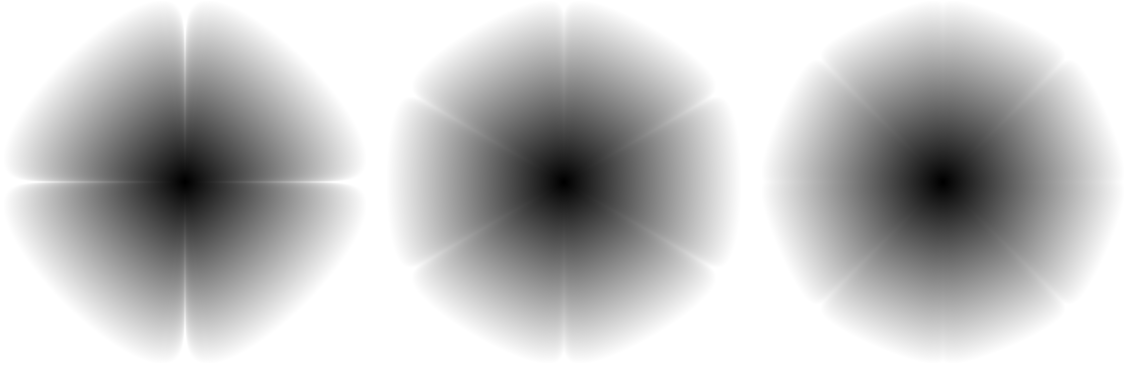


(a) MFD-Quinn, regular (4) (b) MFD-Quinn, hexagonal (c) MFD-Quinn, regular (8)

(d) MFD-Freeman, regular (4) (e) MFD-Freeman, hexagonal (f) MFD-Freeman, regular (8)

(g) MFD-md, regular (4) (h) MFD-md, hexagonal (i) MFD-md, regular (8)

Figure 5.9: Renderings of slope-based weight functions.



(a) Regular (4) (b) Hexagonal (c) Regular (8)

Figure 5.10: Renderings of slope-based weight functions, using optimised values of p.

5.3 Open questions

In this chapter we have managed to prove that for SFD, a circular mountain can better be modelled using a terrain that uses regular grid cells, instead of hexagonal grid cells. For MFD using an angle-based weight function, we have performed measurements that demonstrate the opposite.

In addition, we have done research on slope-based weight functions, where we have seen that existing functions may need to be tuned to provide better performance for specific terrain models.

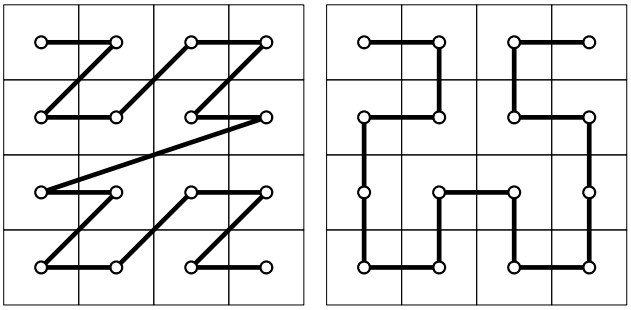
The provided research can be extended in many ways. For example:

- In this chapter we have only used a round hill as an artificial dataset. Are there any other shapes or structures that can be used to compare these terrain models in a fair way?
- So far we have only considered an artificial dataset. How do the results given in this chapter compare to realistic datasets? Are the improved constants for slope-based weight functions for MFD an actual improvement in practice?
- Theoretical analysis has been performed for SFD. Can a similar exact derivation of c_v be given for MFD?
- Research has only been restricted to simulation of flow on two-dimensional terrains. What would be proper measurement criteria for flow simulation in $\mathbb{R}^3$ (e.g. pressure/gas flow simulation)?
- Both the analysed angle-based and slope-based weight functions analyzed suffer from the problem that hairlines occur on the axes of the grid. Which techniques can be used to filter these out effectively?

6 Space-filling curves for hexagonal grid cells

As mentioned in chapter 2, when using a terrain composed of hexagonal grid cells using an integer coordinate system, only half of all possible (x, y)-coordinates can be used to place a hexagon. This means that a naïve approach of simply storing the terrain in two-dimensional array indexed by both axes independently is wasteful. Because a translation of coordinates needs to be performed to prevent this anyway, it is good to look at space-filling curves. An advantage of a space-filling curve is that it may provide better locality of grid cells, which may for some algorithms reduce the number of disk I/Os needed during execution.

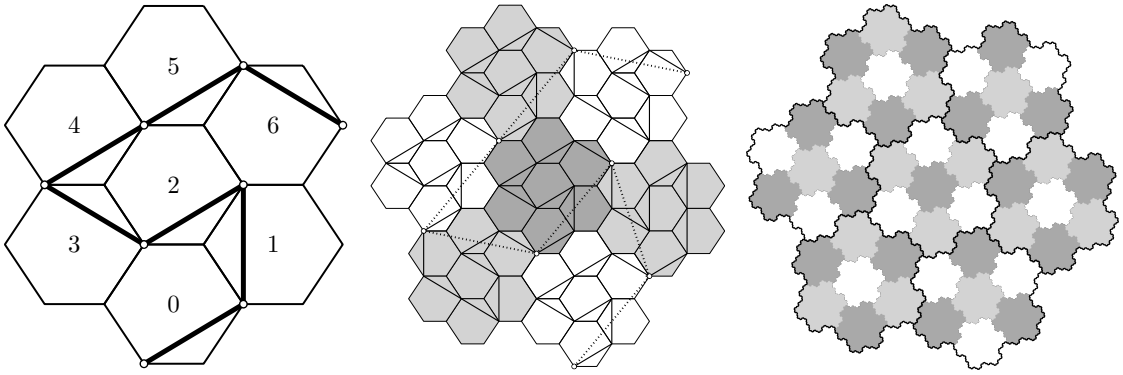
Space-filling curves for regular grids have received much attention in both literature and practice. The most commonly used space-filling curves for regular grids are the Hilbert curve and the Z-order curve. Both of these curves recursively divide a terrain in four regions and traverse through these regions individually. The Z-order curve, shown in Figure 6.1(a), traverses the four regions in a Z-shape. Because it makes discontinuous jumps, it is in most literature considered to be an order – not a curve. The Hilbert curve, shown in Figure 6.1(b), is slightly more complex. It traverses the four regions in a U-shape, with alternating rotations to provide a curve without any discontinuities.



(a) Z-order (b) Hilbert curve

Figure 6.1: Space-filling curves for regular grids.

The Hilbert curve and the Z-order curve are recursive tilings, meaning the shape of a cell is retained at each level of the recursive division. For a terrain with hexagonal grid cells this is not possible; it is not possible to arrange two or more hexagons in such a way that they form a hexagon of greater size.[6] This makes the construction of an algorithm that converts coordinates to curve indices more complicated. A simple top-down conversion is practically infeasible. Because the borders of cells have a very complex structure at higher levels of depth, it cannot easily be determined whether a given coordinate is part of a cell or not.



(a) Depth 1 (b) Depth 2 (c) Depth 21

Figure 6.2: Gosper curves at various levels of depth.

In this chapter we will see there does exist an $O(D)$ time algorithm, which combines a bottom-up, a top-down and a bottom-up pass, where D denotes the depth of the curve. We shall assume that integer arithmetic typically supported by hardware instructions (addition, subtraction, multiplication, division and modulo, but not exponentiation) can be executed in constant time.

6.1 The Gosper curve

A space-filling curve which has been designed for hexagonal grids specifically, is the Gosper curve. This curve, discovered by Bill Gosper in 1973, performs a non-recursive clustering of seven hexagons, as shown in Figure 6.2.[5] The curve enters and leaves each of the hexagons through its vertices. The angle between these two vertices is $\frac{2}{3}\pi$. The seven subcells are not placed at the same angle as the tiling itself. Compared to the tiling, they are placed at an angle of $-\arctan\frac{\sqrt{3}}{5} \approx -0.11\pi$. This rotation can clearly be observed in Figure 6.2(b), by comparing the angle of the thin and the thick curve. At each increasing level of depth, the curve grows by a factor of $\sqrt{7}$ in both dimensions.

The Gosper curve is not the only space-filling curve for hexagonal grids. In fact, the number of space-filling curves based on tilings of hexagons is infinite.[1] In this section we are only considering the case where we have seven hexagons (which we will from now on refer to as $k = 7$). In principle, the described algorithms can be generalised to work with other layouts as well, as long as these layouts are threefold rotational symmetric.

In order to develop algorithms that convert between coordinates and Gosper curve indices, we must first formally define how the hexagons are arranged inside a cell and how the curve goes through them. Not only does each grid cell have an index along the curve, a position in the tiling and an angle at which the curve passes through the cell, we must also consider that some grid cells are traversed in inverse order. For example, consider grid cells 0 and 2 as shown in Figure 6.2(a). Even though these grid cells are rotated equally, the index for points within these subcells must be calculated oppositely. Table 6.1 gives a formal description of each grid cell.

We shall now describe two iterative algorithms that can be used to convert indices along the curve to a pair of (x, y)-coordinates and vice versa. Even though recursive versions of the algorithm are likely to be more simple to analyse, these routines are typically used in critical code paths of a larger application, where iterative versions are more preferable.

Cell	Position	Angle	Inverted
0	$\begin{bmatrix} 0 \\ -2 \end{bmatrix}$	0	no
1	$\begin{bmatrix} 1 \\ -1 \end{bmatrix}$	$-\frac{2}{3}\pi$	yes
2	$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$	0	yes
3	$\begin{bmatrix} -1 \\ -1 \end{bmatrix}$	$\frac{2}{3}\pi$	no
4	$\begin{bmatrix} -1 \\ 1 \end{bmatrix}$	0	no
5	$\begin{bmatrix} 0 \\ 2 \end{bmatrix}$	0	no
6	$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$	$\frac{2}{3}\pi$	yes

Table 6.1: Formal description of grid cells of the Gosper curve.

6.1.1 Converting a curve index to an (x, y)-coordinate

Even though conversion of an (x, y)-coordinate to a curve index is typically performed more often by applications, we start off by describing its inverse function, since it is rather easy to understand and implement.

Before we actually start calculating actual the (x, y)-coordinate at which the curve index is located, we first eliminate the inversion which is induced by three of the grid cells. This transforms the curve into a discontinuous order, but now every cell within the tiling will use the same indexing internally, except for its rotation. Assume the point is placed in a cell at level d which is inverted. We can remove this inversion as follows:

$$
\begin{aligned}
R(i,d) &= \text{last index within cell} - \text{offset within cell} && (6.1)\\
&= \text{first index within cell} + (\text{length of cell} - 1) - \text{offset within cell} && (6.2)\\
&= (i - (i \mod 7^d)) + (7^d - 1) - (i \mod 7^d) && (6.3)\\
&= i + 7^d - 1 - 2 \cdot (i \mod 7^d) && (6.4)
\end{aligned}
$$

We can now apply a bottom-up algorithm, which for each level of depth adds the position of the centrepoint of the cell the curve passes through, while applying rotation to the coordinate calculated at the previous level.

An implementation of this approach is given in Algorithm 6. We shall now prove that the running time of this algorithm is $O(D)$.

Theorem 16 *Converting an index on a Gosper curve, having k cells and depth D, to an (x, y)-coordinate can be calculated in $O(D)$ time. When using the integer-only terrain model, it can be implemented using only integer arithmetic.*

Proof. Looking at the pseudocode provided in Algorithm 6, we can see that both loops separately can be implemented in $O(D)$ time. Assuming it takes $O(1)$ time to determine whether a cell is part of I, R_+ or R_-, each iteration of both loops is bounded by $O(1)$ time. This can for example be accomplished by encoding these sets as bitmasks. The positions of each of the cells, $P(c)$, can be encoded using an array, which also yields $O(1)$ access time. The exponentiations

Algorithm 6 GOSPERINDEXTOCOORDINATE(Integer i)

Let I be the set of cells with inverted subcurves.
Let R_+ be the set of cells having angle $\frac{2}{3}\pi$.
Let R_- be the set of cells having angle $-\frac{2}{3}\pi$.
Let P be a function, mapping cell numbers to their position within the tiling.
{Index must be in bounds.}
if $i < 0 \vee i \geq 7^D$ **then**
    return NIL
{Top-down removal of inverted subcurves.}
for $d \leftarrow D - 1$ **to** 1 **do**
    $c \leftarrow \frac{i}{7^d} \mod 7$
    if $c \in I$ **then**
        $i \leftarrow i + 7^d - 1 - 2 \cdot (i \mod 7^d)$
{Bottom-up calculation of coordinates, taking rotation into account.}
$p \leftarrow \begin{bmatrix} 0 \\ 0 \end{bmatrix}$
$r \leftarrow \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
for $d \leftarrow 0$ **to** $D - 1$ **do**
    $c \leftarrow \frac{i}{7^d} \mod 7$
    {Apply rotation at current level.}
    if $c \in R_+$ **then**
        $p \leftarrow R'(\frac{2}{3}\pi) \cdot p$
    else if $c \in R_-$ **then**
        $p \leftarrow R'(-\frac{2}{3}\pi) \cdot p$
    {Add position of where this cell is placed.}
    $p \leftarrow p + r \cdot P(c)$
    $r \leftarrow \sqrt{7} \cdot R'(\arctan \frac{\sqrt{3}}{5}) \cdot r$
return p

can simply be calculated based on values of previous iterations, meaning all mathematical operations separately can be executed in $O(1)$ time.

To prove that this algorithm can be implemented using only integer math, we need to look at the values assigned to the variables p and r. At any time, the r variable stores the identity matrix, multiplied with $\sqrt{7} \cdot R'(\arctan \frac{\sqrt{3}}{5})$ a certain number of times. Let us first calculate $\sqrt{7} \cdot R'(\arctan \frac{\sqrt{3}}{5})$:

$$\sqrt{7} \cdot R'(\arctan \tfrac{\sqrt{3}}{5}) = \sqrt{7} \cdot \begin{bmatrix} 5/2\sqrt{7} & -1/2\sqrt{7} \\ 3/2\sqrt{7} & 5/2\sqrt{7} \end{bmatrix} \tag{6.5}$$

$$= \begin{bmatrix} 5/2 & -1/2 \\ 3/2 & 5/2 \end{bmatrix} \tag{6.6}$$

As we can see, this matrix has the same property as the rotation matrices we mentioned in chapter 2. As long as the coordinate it is multiplied with conforms to our terrain model, it can perform the transformation using only integer arithmetic. Unfortunately, the columns of the identity matrix do not conform to the terrain model, meaning any multiplication will lead to fractions. This, however, can easily be solved by initialising r with $\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$. This will cause the algorithm to calculate a coordinate having a distance twice as large, but this can simply be solved by returning $\frac{1}{2}p$.

Since the application of r on a point on the terrain will always yield to another point that also conforms to the terrain model, line 24 will always update p to a proper point on the terrain, using only integer arithmetic. □

6.1.2 Converting an (x, y)-coordinate to a curve index

Now that we have described conversion of curve indices to (x, y)-coordinates, we are going to present an algorithm that calculates the inverse. This function is used more often in practice, since applications typically only work with (x, y)-coordinates and simply want to convert these to indices, to determine where to load and store terrain data on storage.

We can already speculate that a top-down approach is not going to help us here. As the level of depth of the curve increases, the border of each of the seven cells is becoming more complex, as shown in Figure 6.2(c). A top-down algorithm would effectively perform point location on a fractal. Similarly, a bottom-up approach is also not of much use. It is impossible to determine in which subcell a point is placed, without knowing how the original cell is positioned and rotated. This depends on information from the levels above.

Let us assume we work with a simplified model of the curve, where none of the cells are inverted or rotated. Would it be possible to create a bottom-up algorithm to perform the conversion? The answer is yes. At the lowest level of the curve, the terrain simply consists of a uniform tiling of seven hexagons. All of these tilings of seven are centered at $\alpha \cdot \begin{bmatrix} 1 \\ -5 \end{bmatrix} + \beta \cdot \begin{bmatrix} 0 \\ 14 \end{bmatrix}$, for some $\alpha, \beta \in \mathbb{Z}$. Since these two vectors are in row echelon form, it is trivial to check whether any given point is a centrepoint of a tiling of seven hexagons. In fact, this specific instance can simply be transformed to checking whether $y + 5x \mod 14 = 0$. Checking whether a given point corresponds with a cell that is not the centrepoint, can simply be accomplished by subtracting the cell's position on beforehand.

Now we have only discussed a way to determine the subcell position at the lowest level of depth. Instead of describing an approach that works for each level separately, we can repeat the previously described mechanism by contracting our terrain, by combining every seven hexagons into one, as shown in Figure 6.3. This can be achieved by multiplying the position of the centrepoint with $\frac{1}{\sqrt{7}} \cdot R'(-\arctan \frac{\sqrt{3}}{5})$. In the end we obtain a list of cells the point is

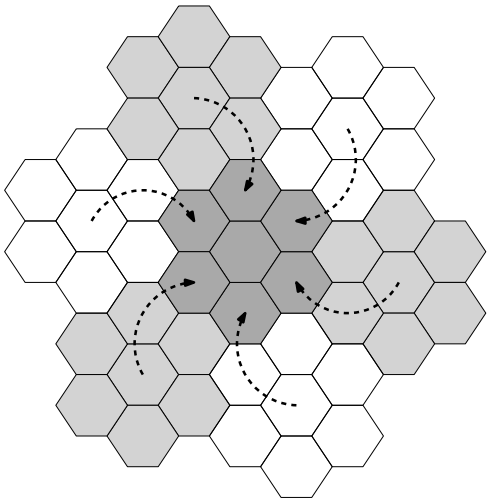


Figure 6.3: Contracting a Gosper curve.

placed in at all the levels of depth. If after executing this procedure D times, the point still has a non-zero coordinate, it means that the point is not placed within a Gosper curve of size k^D.

Now that we have found a procedure that allows us to determine in which cell a point is placed at all levels, we can use a top-down traversal to apply the rotation and a bottom-up traversal to restore the inversion. This is the opposite of what is done in the algorithm that maps curve indices to (x, y)-coordinates.

Theorem 17 *Converting an (x, y)-coordinate to an index on a Gosper curve having k cells and depth D, can be calculated in $O(kD)$ time. When using the integer-only terrain model, it can be implemented using only integer arithmetic.*

Proof. Looking at the pseudocode shown in Algorithm 7, we can conclude that the first pass of the algorithm requires $O(kD)$ time. The second and third pass of the algorithm require $O(D)$ time.

Line 23 of the algorithm depends on an inverse function that is able to map points to cell indices. We can still perform this in constant time, if we simply replace it by a new a function:

$$P^*(c, \varphi) \quad = \quad P^{-1}(R'(\varphi) \cdot P(c)) \tag{6.7}$$

Because φ can only have three possible values, 0, $\frac{2}{3}\pi$ and $-\frac{2}{3}\pi$, this function can be implemented by using three separate lookup tables, meaning it can be executed in constant time.

To determine that the algorithm works properly with integer arithmetic only, we calculate the rotation matrix that is used on line 14:

$$\tfrac{1}{\sqrt{7}} \cdot R'(-\arctan \tfrac{\sqrt{3}}{5}) \quad = \quad \tfrac{1}{\sqrt{7}} \cdot \begin{bmatrix} 5/2\sqrt{7} & 1/2\sqrt{7} \\ -3/2\sqrt{7} & 5/2\sqrt{7} \end{bmatrix} \tag{6.8}$$

$$= \quad \begin{bmatrix} 5/14 & 1/14 \\ -3/14 & 5/14 \end{bmatrix} \tag{6.9}$$

When multiplying this rotation matrix with a vector – as is done on line 14 – we can perform the division by 14 after performing the addition first. We can say that this will not cause any rounding in the end result, since p_c always denotes a centrepoint of a cluster of seven hexagons. Every centrepoint has a corresponding cell on the grid after contraction. □

Algorithm 7 GOSPERCOORDINATETOINDEX(Coordinate p)

{Respect terrain model.}
if $p.x \mod 2 \neq p.y \mod 2$ **then**
    return NIL
{Bottom-up calculation of cell index, not taking rotation and inversion into account.}
Let C be a stack to store cell numbers.
for $d \leftarrow 1$ **to** $D - 1$ **do**
    {Determine in which cell p is placed at the root level.}
    for $c \leftarrow 0$ **to** 6 **do**
        $p_c \leftarrow p - P(c)$
        if $(p_c.y + 5 \cdot p_c.x) \mod 14 = 0$ **then**
            break
    {Store at which cell p is placed and rotate terrain inwards.}
    PUSH(C, c)
    $p \leftarrow \frac{1}{\sqrt{7}} \cdot R'(-\arctan\frac{\sqrt{3}}{5}) \cdot p_c$
{Check whether coordinate is inside the Gosper island.}
if $p \neq \begin{bmatrix}0\\0\end{bmatrix}$ **then**
    return NIL
{Top-down application of rotation.}
$\varphi \leftarrow 0$
$i \leftarrow 0$
for $d = D - 1$ **to** 1 **do**
    {Obtain cell index, taking rotation into account.}
    $c \leftarrow P^{-1}(R'(\varphi) \cdot P(\text{POP}(C)))$
    $i \leftarrow i + c \cdot 7^d$
    if $c \in R_+$ **then**
        $\varphi \leftarrow \varphi - \frac{2}{3}\pi$
    else if $c \in R_-$ **then**
        $\varphi \leftarrow \varphi + \frac{2}{3}\pi$
{Bottom-up reintroduction of inverted subcurves.}
for $d \leftarrow 1$ **to** $D - 1$ **do**
    $c \leftarrow \frac{i}{7^d} \mod 7$
    if $c \in I$ **then**
        $i \leftarrow i + 7^d - 1 - 2 \cdot (i \mod 7^d)$
return i

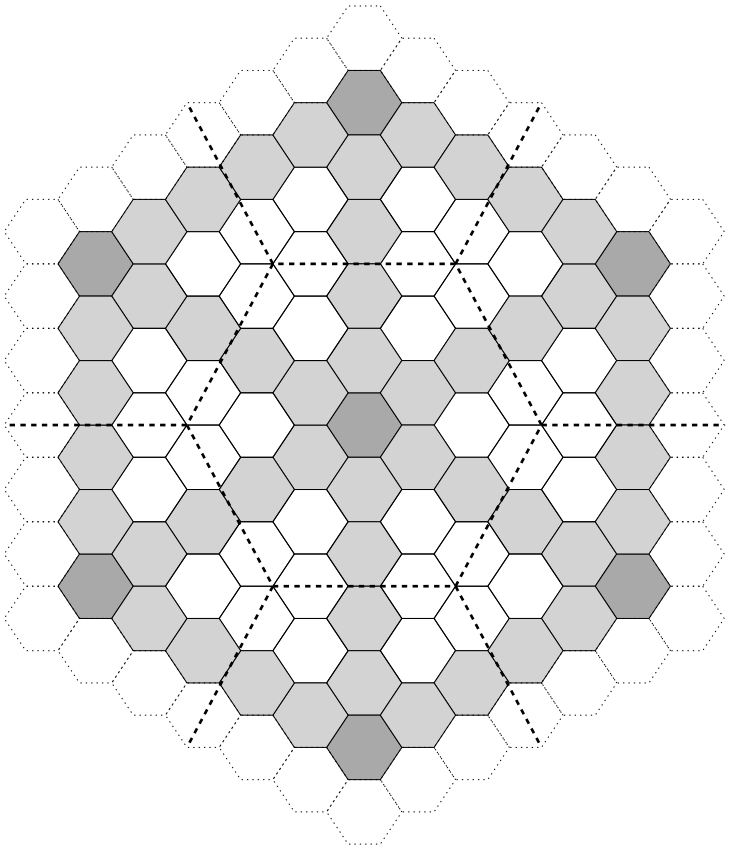


Figure 6.4: A tiling where $i = 2$.

It should be noted that for this specific curve, we can use a small trick to improve its running time to $O(D)$. Probably by coincidence, the formula $y + 5x \mod 7$ returns a distinct number for every of the seven cells within the tiling. Instead of testing each of the cells separately, we can simply use this formula, in combination with a lookup table to obtain the cell number in $O(1)$ time.

6.2 Constructing a rotation-free space-filling curve

Even though the previously discussed Gosper curve often suffices the requirements of providing a practically usable space-filling curve for hexagonal grids, its shape does have a disadvantage; it doesn't allow the terrain to be scaled easily. Consider that one wants to increase the resolution of an existing terrain, having 7^D cells stored on a Gosper curve of depth D. One easy way to accomplish this, is to replace each hexagon with a cell of seven hexagons. Such a terrain can be stored on a Gosper curve of depth $D + 1$, but unfortunately this transformation rotates the terrain with an angle of $\arctan \frac{\sqrt{3}}{5}$.

In this section we are going to construct a new space-filling curve of which the tiling is sixfold rotational symmetrical and is rotation-free. In this sense, rotation-free means that curve runs in such a way that the angles at which the entry point and the exit point of the curve are placed with respect to the centrepoint, only differ by a multiple of $\frac{2}{3}\pi$ between levels of depth.

Lemma 18 *Every rotation-free space-filling curve having a sixfold rotational symmetrical tiling of hexagonal grid cells has $k = (2i + 1)^2$ cells, where $2i + 1$ denotes the distance between the centrepoints of two adjacent tilings.*

Proof. Sixfold rotational symmetry for a tiling of hexagons can only be achieved by placing the centrepoint of the tiling in the middle of the face of a hexagon. Placing the centrepoint on an edge or vertex only yields twofold or threefold symmetry. Also, because the space-filling curve has to be rotation-free, the shape of the tiling is restricted to the layout shown in Figure 6.4. In this figure, the dark grey hexagon denotes the centrepoint and the light grey hexagons denote the legs that connect the centrepoints of adjacent tilings, having length i.

In addition to its centrepoint and the $6i$ hexagons that connect the centrepoints of the tilings, each tiling also has to pick cells from the white pockets that reside between three centrepoints. Each of these pockets has size $\sum_{j=0}^{2i-1} j$. To maintain sixfold rotational symmetry, each tiling can only pick $\frac{1}{3}$ of cells from each pocket.

This means that a tiling where the centrepoints have a distance of $2i+1$ has the following number of tiles:

$$\begin{aligned} k &= 1 + 6i + 6 \cdot \tfrac{1}{3} \sum_{j=0}^{2i-1} j && (6.10)\\ &= 1 + 6i + 2 \cdot \tfrac{1}{2} \cdot 2i(2i-1) && (6.11)\\ &= (2i+1)^2 && (6.12) \end{aligned}$$

□

Now we know that we only have to consider tilings of size $k = 9, 25, 49, 81, 121, ...$ Unfortunately, we have to make the following observation:

Lemma 19 *There exists no rotation-free space-filling curve, having sixfold rotational symmetry, where $k < 49$.*

Proof. First, we consider the case where $i = 1$, thus $k = 9$. In this case, we know that each of the pockets can only consist of $\frac{9-1-6}{6} = \frac{1}{3}$ cells. This is not a valid integer.

Now we consider the case where $i = 2$, thus $k = 25$. For $i = 2$, we have to pick $\frac{25-1-12}{6} = 2$ cells from every corner pocket. If a tiling has a cell with only one neighbour (thus at least six cells having one neighbour), it is impossible to calculate a Hamiltonian path – let alone a valid space-filling curve. This means that there is only one way we can create a rotation-free sixfold rotational symmetric tiling where $k = 25$, namely as shown in Figure 6.5.

Because the curve enforces that only three of the six vertices of each hexagon are used, we can assume without loss of generality that the curve only uses the paths shown in black. Then there are three hexagons which share only a single vertex of the curve with the others, as shown in grey. These cells cannot be passed through, meaning they must either be the first or the last cell on the curve. Of course this cannot be realised with more than two cells.

Since 9 and 25 were our first two candidates, we can conclude that there cannot exist a rotation-free space-filling curve, having sixfold rotational symmetry, smaller than 49. □

Theorem 20 *The smallest rotation-free space-filling curve, having a sixfold rotational symmetric tiling of hexagonal grid cells has size 49.*

Proof. Combining lemmas 18 and 19, we can conclude that the next candidate tiling has size 49. To demonstrate this, we have generated two curves, which we shall call the alternating Gosper curve and the Marita curve, as shown in Figure 6.6.

The alternating Gosper curve is similar to the original Gosper curve. Essentially it has the same pattern as a Gosper curve of depth 2, except that between levels, the curve is applied in inverse order. This effectively cancels out the $\arctan \frac{\sqrt{3}}{5}$ angle.

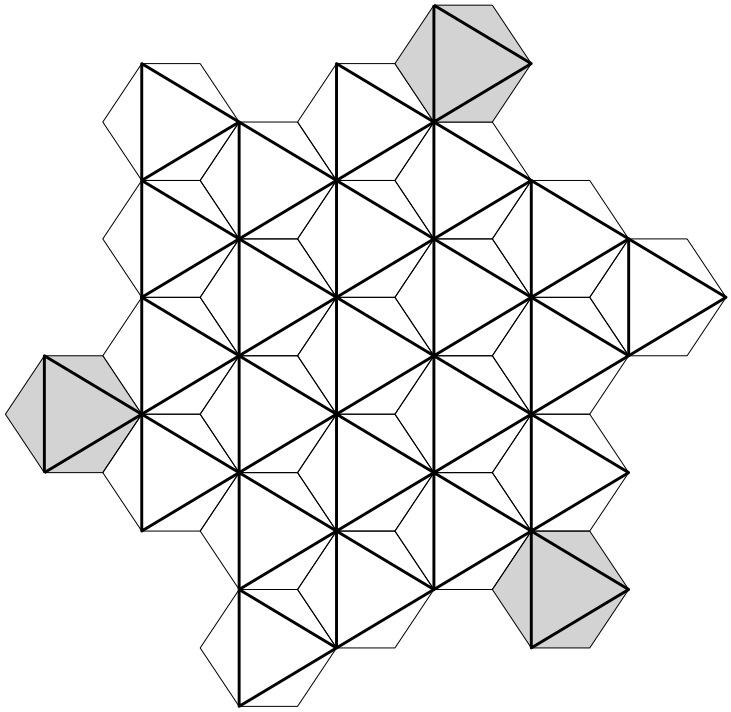


Figure 6.5: Only tiling of size 25, where all cells have two or more neighbours.

The Marita curve uses the same tiling, but uses a curve that differs for 12 of the 49 cells. The advantage of this curve compared to the alternating Gosper curve, is that the 25th cell on the curve corresponds with the centrepoint of the tiling. If one creates a curve of higher depth using this approach, point $\begin{bmatrix}0 & 0\end{bmatrix}$ will always reside at index $\lfloor \frac{1}{2}k^D \rfloor$. By normalising the curve indices to range from $-\lfloor \frac{1}{2}k^D \rfloor$ to $\lfloor \frac{1}{2}k^D \rfloor$, one can create a space-filling curve of practically infinite size where indices grow automatically as a larger portion of the terrain is used.

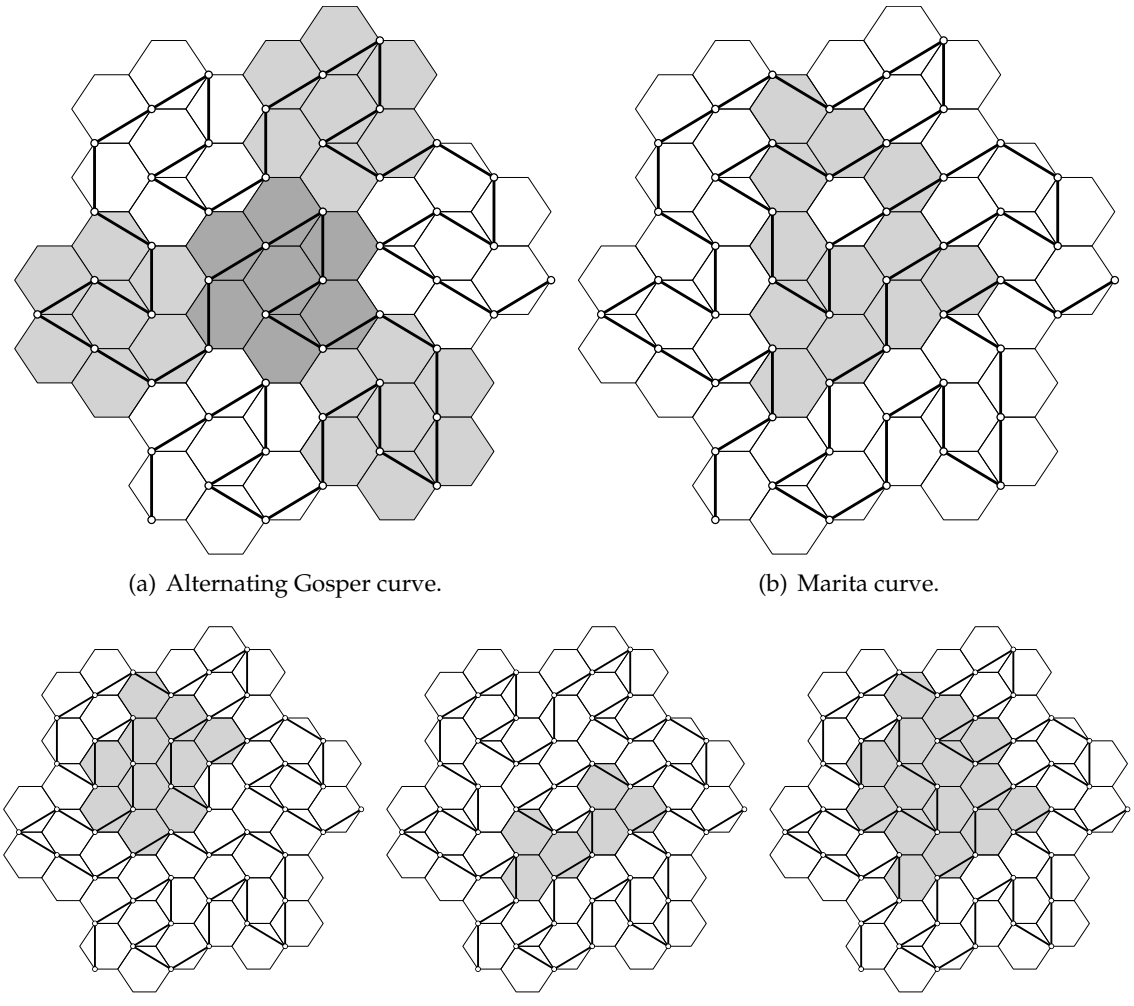
In addition to the two curves mentioned previously, this tiling has three other curves that are rotation-free. □

6.3 Adapting the algorithms to work with rotation-free tilings

The algorithms described previously can easily be adjusted to allow conversion of indices and (x, y)-coordinates of the rotation-free space-filling curves, having sixfold rotational symmetry. Unfortunately, the running time of GOSPERCOORDINATETOINDEX is $O(kD)$, while we mentioned an optimisation for Gosper curves specifically that reduces the running time to $O(D)$. Would it be possible to do something similar for these curves as well?

Essentially, we have to develop a procedure that maps (x, y)-coordinates to cell numbers for a uniform tiling at the lowest level of the curve. The sixfold rotational symmetric tiling is pretty hard to reason about, so let us assume we are performing a uniform tiling of the terrain using a different shape, namely a rhombus of size $(2i + 1)^2$.

Assume we use a simple way of indexing the cells, namely lexicographically by row and column, as shown in Figure 6.7. Also assuming there is a rhombus for which cell 0 is placed at $\begin{bmatrix}0 & 0\end{bmatrix}$, we can easily convert a coordinate to the column it corresponds to, using $x \mod (2i + 1)$. The row it is placed in can be calculated using $\frac{y-x}{2} \mod (2i + 1)$. This means the cell number can be derived as follows:



(a) Alternating Gosper curve. (b) Marita curve.

(c) Unnamed curve 1. (d) Unnamed curve 2. (e) Unnamed curve 3.

Figure 6.6: Rotation-free space-filling curves of size 49. Grey cells indicate difference with respect to the Alternating Gosper curve.

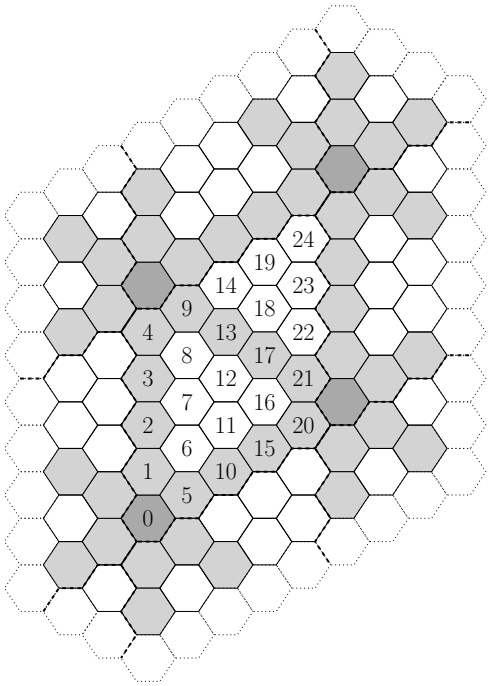


Figure 6.7: A tiling of rhombuses of size $(2i+1)^2$, where $i = 2$.

$$C(p) \quad = \quad (\tfrac{p.y-p.x}{2} \mod (2i+1)) + (p.x \mod (2i+1)) \cdot (2i+1) \tag{6.13}$$

Now we can make the following interesting observation. The repetition of rhombuses of size $(2i+1)^2$ on the plane is identical to the repetition of sixfold rotational symmetric tilings of equal size. This means that there exists a bijection between its cell numbers. This allows us to simply map the result of $C(p)$ to correspond with the actual cell number of the sixfold rotational symmetric tiling, using constant time.

The fact that this is possible leads to a rather special property. We have now constructed a framework where the running time of the algorithms is independent of k. This means that increasing k could in theory make these procedures faster, since a lower value of D can be used to store the same number of points on the curve.

6.4 Performance of conversion algorithms

Even though theoretical analysis of the conversion algorithms is useful, in typical cases the values k and D are small enough that theoretical upper bounds provide no practical value. This is why it is more interesting to look at running times of an actual implementation.

Both the algorithm to convert indices to (x, y)-coordinates and its counterpart have been implemented in the C programming language and have been released to the community under the 2-clause BSD license[1]. To provide a frame of reference, these algorithms will be compared against an implementation of conversion routines for the Hilbert curve, obtained from Wikipedia[2].

The largest Alternating Gosper curve and Marita curve that can be addressed using 64-bit indices and a 32-bit coordinate model, has size 49^{10}. Therefore, we shall compare it against a

[1] Flowsnake: `http://80386.nl/projects/flowsnake/`
[2] Hilbert curve: `http://en.wikipedia.org/wiki/Hilbert_curve`

	Gosper curve	Alternating Gosper curve Marita curve	Hilbert curve
Curve length	$7^{20} \approx 8.0 \cdot 10^{16}$	$49^{10} \approx 8.0 \cdot 10^{16}$	$4^{28} \approx 7.2 \cdot 10^{16}$
Object file size	1572 bytes	1687 bytes	354 bytes
Index to (x, y)-coordinate	1653 ns	561 ns	240 ns
(x, y)-coordinate to index	1528 ns	768 ns	210 ns

Table 6.2: Performance of conversion routines.

Gosper curve of size 7^{20}. The Hilbert curve uses a tiling of size 4. When choosing size 4^{28}, the curve has 90% the size of the other curves.

We simply benchmark the algorithms by invoking them 10^{10} times and dividing the total running time of the benchmark process by the number of invocations. For the index-to-coordinate conversion, we select a chain of length 10^{10} passing through to the centre of the grid. For the coordinate-to-index conversion, we select a rectangular area of size 4000×5000, placed at the centre of the grid.

When executing this benchmark on an Intel Core 2 Duo 2.26 GHz running Ubuntu 11.04, shipped with GCC 4.5.2, we gain the results shown in Table 6.2. We can immediately conclude that the algorithms for hexagonal grids have a hard time competing with algorithms for regular grids. Not only are the algorithms bigger in code size, they are also slower. This is to be expected, considering that our algorithms are a lot more complex. Whereas our algorithms require 2 or 3 passes to calculate its final result, the Hilbert curve can be processed with a single pass.

As we analysed, the running time of our algorithms does not depend on the size of the tiling, but only on the depth of the curve. This can clearly be observed by comparing the results for the Gosper curve and the rotation-free curves. While we see a twofold performance improvement for coordinate-to-index conversion, the performance of index-to-coordinate conversion has increased a threefold. This threefold performance improvement is likely due to the fact that the lack of rotation allows the compiler to simplify the arithmetic significantly. The compiler can optimize the rotation matrix to a single scalar value.

Even though the use of space-filling curves for hexagonal grids comes at a larger cost than it does for regular grids, it may well be the case that this amount of overhead is acceptable. In our measurements we used rather large curve sizes. Considering we want to map the surface of the Earth on a Gosper curve of depth 20, we would have $\frac{7^{20}}{5.1 \cdot 10^{14}} \approx 156$ grid cells per square metre. In almost all practical cases, a lower level of depth is sufficient. Still, at a depth of 20, we are capable of performing conversion in less than a microsecond. Even in cases where disk I/Os only have a millisecond of latency, this approach already runs break even when it can prevent a single disk I/O every 1000 calls.

6.5 Open questions

Even though this chapter already provides insight in complex ways of applying space-filling curves to a terrain of hexagonal grid cells, there are still many topics that are open for research.

- The algorithms provided in this chapter are specifically designed for hexagonal grid cells in $\mathbb{R}^2$. Can they be generalised for a larger class of terrain models, for example tilings of bitruncated cubes in $\mathbb{R}^3$? If so, can a running time of $O(D)$ still be realised?
- As proven, the smallest sixfold rotational symmetrical rotation-free tiling is composed of 49 hexagons. Would it be possible to create smaller rotation-free tilings that are only threefold rotational symmetric?

7 Terrains with variable-sized grid cells

In earlier chapters, we have compared performance of various terrain models under the assumption that all grid cells are identical in shape and size. Though often easy to implement, it places a strong requirement on the sampling resolution.

- If the terrain is sampled under a resolution that is too large, the grid may encode a large amount of redundant information, for example by encoding a flat surface using a large number of samples that are equal in value.
- Even worse, if the resolution is too low, the stored terrain might lose its original semantics.

For large environments this can easily cause a dilemma. Certain parts of a terrain may require a large amount of detail to encode properly, whereas other parts of the terrain have low complexity and will suffer from redundancy. To provide a good trade-off between detail and storage space requirements, one could implement an adaptive approach, where the resolution can be differed for parts of the terrain as needed.

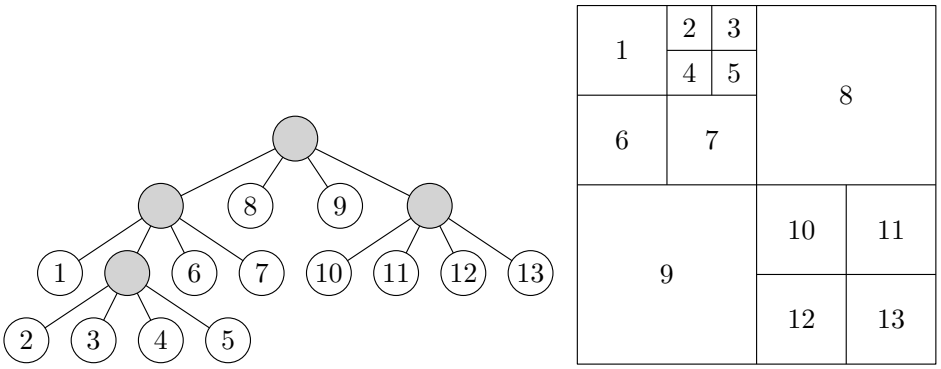
In this chapter we are going to discuss two data structures for storing terrains using adaptive grid cell sizes, namely quadtrees and septuatrees. As septuatrees seem not to be published in existing literature, we shall also describe how various fundamental operations on these trees can be implemented, which are used by practically any algorithm that operates on these data structures.

Finally, we shall compare their performance by repeating the benchmarks for least-cost paths as described in chapter 4.

7.1 Quadtrees for regular grid cells

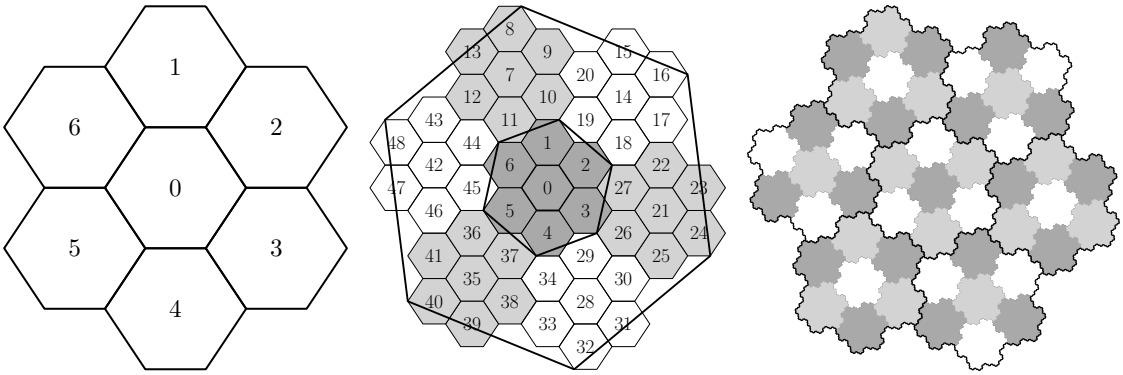
Likely the most common technique to store grid terrains using variable precision, is by storing the grid in a quadtree, as characterised by Finkel and Bentley[3]. A quadtree is a tree structure that is composed of two types of nodes:

- Leaf nodes, which store a value for a square piece of the terrain. More specifically, if a leaf is stored at height h, it corresponds with a tile of size $2^h \times 2^h$.
- Non-leaf nodes, which when stored at height h, decomposes a tile of size $2^h \times 2^h$ into four tiles of size $2^{h-1} \times 2^{h-1}$. A non-leaf node has exactly four children.



(a) Tree representation (b) Grid representation

Figure 7.1: A simple quadtree.



(a) Depth 1 (b) Depth 2 (c) Depth 21

Figure 7.2: G-order at various levels of depth.

Because of this structure, a quadtree always has $1 + 3k$ leaves, where k is the number of non-leaf nodes. In this document we shall assume the four children are always stored in Z-order. Figure 7.1 shows an example of a simple quadtree, storing a terrain of size 8×8, dividing 64 points over 13 grid cells.

7.2 Septuatrees for hexagonal grid cells

Now that we have described a method of storing adaptive grid cells ordered in Z-order in a tree structure, we can mimic the same approach to construct a tree for hexagons, based on the tiling of the Gosper curve. As each non-leaf node in this tree has exactly seven children, we shall call this tree a *septuatree*.

As we've seen in the previous chapter, implementing algorithms that operate on Gosper curves can be a tedious job. Just as the Hilbert curve and Z-order only differ in their ordering and not in their shape, we shall therefore base the work in this chapter on an easier layout, which we call the G-order. Instead of using the cell description provided by Table 6.1, it uses a simpler layout as shown in Table 7.1. Figure 7.2 depicts renderings of this order.

In order to execute algorithms on top of septuatrees, we must first describe how primitive operations that are used by these algorithms can be implemented. First, we shall look at operations that can be used to convert between coordinates on the grid and nodes in the tree. Then we shall define an algorithm that computes the set of neighbours for a given grid cell. Such an operation is fundamental for implementing a least-cost path algorithm efficiently.

Cell	Position	Angle	Inverted
0	$\begin{bmatrix}0\\0\end{bmatrix}$	0	no
1	$\begin{bmatrix}0\\2\end{bmatrix}$	0	no
2	$\begin{bmatrix}1\\1\end{bmatrix}$	0	no
3	$\begin{bmatrix}1\\-1\end{bmatrix}$	0	no
4	$\begin{bmatrix}0\\-2\end{bmatrix}$	0	no
5	$\begin{bmatrix}-1\\-1\end{bmatrix}$	0	no
6	$\begin{bmatrix}-1\\1\end{bmatrix}$	0	no

Table 7.1: Formal description of grid cells of the G-order.

7.2.1 Converting a septuatree node to an (x, y)-coordinate

For septuatrees, node-to-coordinate conversion can simply be implemented by performing a bottom-up traversal of the node given. At each level, we add to the result the placement of the node with respect to the position of the parent. Algorithm 8 demonstrates how such a procedure can be implemented for septuatrees.

Algorithm 8 SEPTUATREENODETOCOORDINATE(Node n)

```
1: Let P be a function, mapping cell numbers to their position within the tiling.
2: {Bottom-up calculation of coordinates.}
3: p ← [0 0]
4: while PARENT(n) ≠ NIL do
5:         for c ← 0 to 6 do
6:                 if CHILD(PARENT(n), c) = n then
7:                         {Node is part of the c'th child.}
8:                         break
9:         {Add position of where this cell is placed.}
10:        p ← p + (√7 · R'(arctan √3/5))^HEIGHT(n) · P(c)
11:        n ← PARENT(n)
12: return p
```

Theorem 21 *The* SEPTUATREENODETOCOORDINATE *algorithm runs in $O(H)$ worst-case time, under the assumption that integer arithmetic can be executed in $O(1)$ worst-case time. H denotes the height of the septuatree.*

Proof. The loop on line 4 is executed at most H times and disregarding line 10, each iteration takes $O(1)$ time. The result of the exponentiation performed on line 10 can be reused between iterations, as the exponent increases by 1 between every iteration. As the exponent is at most equal to H, line 10 can be implemented in $O(H)$ time over all iterations. □

7.2.2 Converting an (x, y)-coordinate to a septuatree node

As we saw with the Gosper curve, it is not possible to convert coordinates to indices on the curve using a single top-down pass. Because the G-order uses the same tiling as the Gosper curve, it is therefore also impossible to convert a coordinate to a node at once.

Therefore we implement coordinate-to-node conversion using multiple passes. Just like the conversion routine for the Gosper curve, we use the first pass to obtain the cell numbers bottom-up. The second pass is then used to walk down the septuatree to obtain the node in the tree.

Algorithm 9 implements coordinate-to-node conversion, based on many of the principles of the coordinate-to-index conversion algorithm for Gosper curves. To provide flexibility, this algorithm can also be used to return a subtree in which a point resides. The m argument denotes the maximum depth of the node returned. If $m = \infty$, the algorithm is guaranteed to return a leaf node.

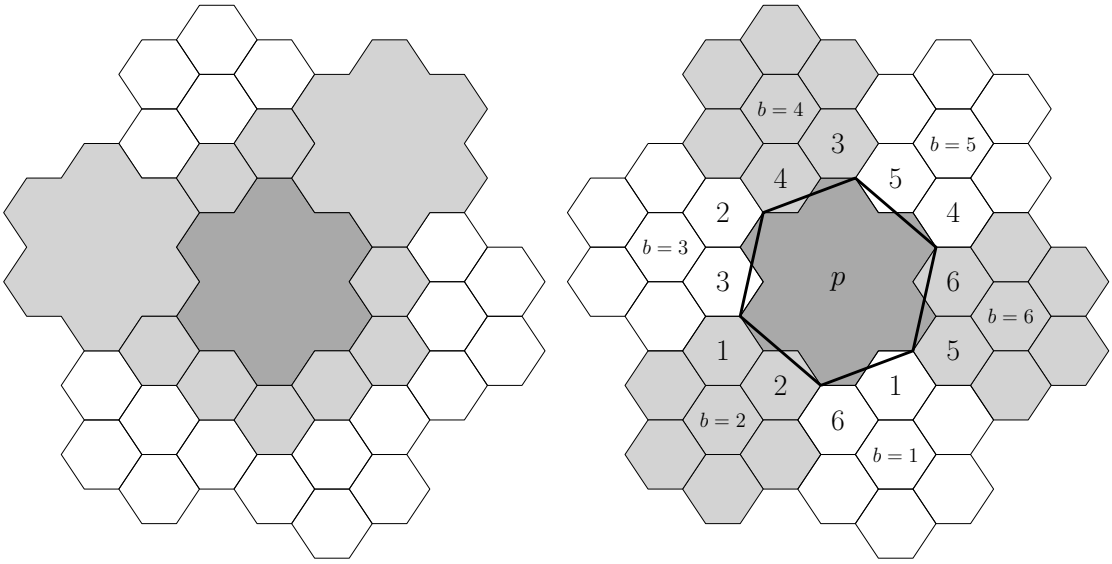
Algorithm 9 SEPTUATREECOORDINATETONODE(Coordinate p, Tree T, Integer m)

1: Let P be a function, mapping cell numbers to their position within the tiling.
2: {Respect terrain model.}
3: **if** $p.x \mod 2 \neq p.y \mod 2$ **then**
4: **return** NIL
5: {Bottom-up calculation of cell number in tree.}
6: Let C be a stack to store cell numbers.
7: **for** $d \leftarrow 1$ **to** HEIGHT$(T) - 1$ **do**
8: **for** $c \leftarrow 0$ **to** 6 **do**
9: $p_c \leftarrow p - P(c)$
10: **if** $(p_c.y + 5 \cdot p_c.x) \mod 14 = 0$ **then**
11: **break**
12: {Store at which cell p is placed and rotate terrain inwards.}
13: PUSH(C, c)
14: $p \leftarrow \frac{1}{\sqrt{7}} \cdot R'(-\arctan \frac{\sqrt{3}}{5}) \cdot p_c$
15: {Check whether coordinate is inside the snowflake.}
16: **if** $p \neq \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ **then**
17: **return** NIL
18: {Top-down traversal of tree to obtain node.}
19: $n \leftarrow$ ROOT(T)
20: **while** $\neg$ISLEAF$(n) \wedge m > 0$ **do**
21: $n \leftarrow$ CHILD$(n,$ POP$(C))$
22: $m \leftarrow m - 1$
23: **return** n

Compared to a quadtree, we see that the non-recursive structure of the tiling requires us to first convert the coordinate into a sequence of steps in the septuatree bottom-up. Whereas on the quadtree such an algorithm can run in $O(\min(H, m))$ time, this cannot be achieved for septuatrees.

Theorem 22 *The* SEPTUATREECOORDINATETONODE *algorithm has a worst-case running time of $O(H)$ on the assumption that integer arithmetic can be performed in $O(1)$ time, where H denotes the depth of the septuatree.*

Proof. The loop performed on line 7 of the algorithm runs at most $H - 1$ iterations, where each iteration can be executed in $O(1)$ time. The second loop either terminates when $m = 0$



(a) This dark grey leaf has 10 neighbouring leaves. (b) Grid cells on boundaries of neighbouring nodes.

Figure 7.3: Neighbours in septuatrees.

or when a leaf node is obtained, therefore having a worst-case running time of $O(\min(H, m))$. This leads to a total worst-case running time of $O(H)$. □

7.2.3 Neighbour queries

When implementing a least-cost path algorithm for grids, it is of great importance that we can efficiently obtain a set of neighbouring grid cells for a given node in the tree, as shown in Figure 7.3(a). For recursive tilings such as the quadtree, these operations are rather intuitive and can even be solved at the cost of poor performance by querying every coordinate on the outer boundary of the grid cell. For the septuatree this is not as easy, as the tiling is not recursive and the boundary of a cell thus has a complex shape. Still it is possible to implement neighbour queries on septuatrees using an algorithm that is structurally identical to existing algorithms for quadtrees.

A node p in a septuatree is enclosed by up to six neighbouring nodes, which either have the same depth as node p, or are leaves of lower depth. To obtain the set of neighbouring grid cells, we can traverse each of these neighbouring nodes to return its cells on the outer boundary adjacent to node p.

First, let us first define a recursive function that traverses all leaves of neighbouring node n for a given boundary on edge b of that node. If the node is a leaf, then we can simply return node n. If the node is not a leaf, we must traverse its children. Depending on the value of b, we must choose a different set of children.

As illustrated in Figure 7.3(b), every non-leaf neighbour node n is connected to the node p by two of its child nodes. For example, the white southernmost neighbour is connected to node p through cells 1 and 6. Because cell 1 shares two edges with node p, we must traverse this child twice. For all six neighbours, this leads to the recursion scheme on the child nodes as provided in Table 7.2.

Using this table, we can implement boundary traversal as shown in Algorithm 10. Instead of implementing all six cases of b explicitly, it uses the PREVIOUSNEIGHBOUR routine to obtain the previous neighbouring cell on the boundary (e.g. by computing $(b + 4) \mod 6 + 1$).

Now that we have a routine that can traverse the boundary of a single neighbour, we can

b	Cell 1	Direction 1	Cell 2	Direction 2	Cell 3	Direction 3
1	1	1	1	6	6	1
2	2	2	2	1	1	2
3	3	3	3	2	2	3
4	4	4	4	3	3	4
5	5	5	5	4	4	5
6	6	6	6	5	5	6

Table 7.2: Children that need to be recursed for a given value of b.

Algorithm 10 SEPTUATREETRAVERSEBOUNDARY(Node n, Integer b)

if ISLEAF(n) **then**
    {Node is leaf. Return it directly.}
    return n
else
    {Node is non-leaf. Traverse boundaries of its three edges.}
    $r_1 \leftarrow$ SEPTUATREETRAVERSEBOUNDARY(CHILD(n, b), b)
    $r_2 \leftarrow$ SEPTUATREETRAVERSEBOUNDARY(CHILD(n, b), PREVIOUSNEIGHBOUR(b))
    $r_3 \leftarrow$ SEPTUATREETRAVERSEBOUNDARY(CHILD(n, PREVIOUSNEIGHBOUR(b)), b)
    return $r_1 + r_2 + r_3$

compute the entire set of neighbours for node p by executing it six times, as demonstrated in Algorithm 11. It uses the existing node-to-coordinate and coordinate-to-node routines to obtain the corresponding neighbouring nodes n to perform boundary traversal on. As we must traverse the boundary of the neighbour adjacent to node p, the code uses an OPPOSITENEIGHBOUR function to convert the boundary number (e.g. by computing $(b + 2) \mod 6 + 1$).

Algorithm 11 SEPTUATREERETURNNEIGHBOURS(Node p)

Let P be a function, mapping cell numbers to their position within the tiling.
$r \leftarrow \emptyset$
for $c \leftarrow 1$ **to** 6 **do**
    {Calculate coordinate of centrepoint of neighbour.}
    $l \leftarrow$ SEPTUATREENODETOCOORDINATE(p) + $(\sqrt{7} \cdot R'(\arctan \frac{\sqrt{3}}{5}))^{\text{HEIGHT}(n)} \cdot P(c)$
    {Top-down traversal of neighbour along its boundary.}
    $n \leftarrow$ SEPTUATREECOORDINATETONODE(l, TREEOFNODE(p), DEPTH(m))
    if $n \neq$ NIL **then**
        $r \leftarrow r +$ SEPTUATREETRAVERSEBOUNDARY(n, OPPOSITENEIGHBOUR(c))
return r

Now that we have presented an algorithm to compute the set of neighbours for a node in a septuatree, we shall derive its running time.

Theorem 23 *The worst-case running time of* SEPTUATREERETURNNEIGHBOURS *is* $O(H + k)$*, where* k *is the number of leaves reported, assuming integer arithmetic can be performed in* $O(1)$ *time.*

Proof. Disregarding the boundary traversal, the SEPTUATREERETURNNEIGHBOURS function runs in $O(H)$ worst case time, as it calls SEPTUATREENODETOCOORDINATE and SEPTUATREECOORDINATETONODE a constant number of times. Also the exponent used to calculate the coordinates of the neighbours can be computed in $O(H)$ time.

The worst-case running time of the invocation of SEPTUATREETRAVERSEBOUNDARY can be derived as follows. Every leaf node of the tree returned by this function is traversed for every edge it shared with node p. As each cell on the boundary of the tiling has exactly three neighbours, each grid cell has at most three edges on the boundary of the tiling. This means that SEPTUATREETRAVERSEBOUNDARY is invoked on at most $3k$ leaf nodes of the septuatree.

As we know the number of traversed leaf nodes, we can now use the following recurrence to derive the total number of invocations of SEPTUATREETRAVERSEBOUNDARY:

$$\begin{aligned} T(k) &= 3 \cdot \sum_{i=0}^{H} \frac{k}{3^i} && (7.1) \\ &< 3k \cdot \sum_{i=0}^{\infty} \frac{1}{3^i} && (7.2) \\ &= 4.5k && (7.3) \end{aligned}$$

As each invocation separately consumes $O(1)$ time, the worst-case running time of SEPTUATREERETURNNEIGHBOURS therefore is $O(H + k)$. □

7.3 Measuring least-cost path quality

Now that we have described how quadtrees and septuatrees work, we shall repeat the measurements on least-cost path quality that were performed in section 4.4. During every measurement we convert the same TIN of 125 vertices to 360 grids. Each of these grids is sampled from the TIN under a different angle, in 1° steps.

For these 360 grids, we calculate 360 shortest paths from the centrepoint of the grid to uniformly spaced points on a ring at $\frac{3}{4}$ the radius of the TIN. These 360×360 least-cost paths are combined to obtain the coefficient of variance.

Before computing these least-cost paths, the grids are converted to quadtrees and septuatrees. These trees are simplified iteratively by repeatedly replacing a node with seven leaf nodes as its children by a new leaf node. The cost value of the new leaf node is equal to the average value of the children. During each iteration, the node selected to be replaced is the one that would cause the lowest per-cell difference in weight to be added to the terrain.

Least-cost paths are calculated in two different ways:

- by converting the septuatree back to a uniform grid and executing the original least-cost path algorithms, and
- by calculating the least-cost paths on the cells of the tree structure directly.

The first approach is far more precise than the second approach, as it allows the path to traverse the grid at a higher level of precision. For the first approach, we only consider the tree structure to be an intermediate storage format for the terrain. When using the second approach, a sparsely filled part of the terrain can only be traversed by performing large steps, thus likely causing detours.

	Resolution of grid						
Cells in tree	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}	2^{22}
2^{10}	0.024863	0.017518	0.017931	0.018831	0.019316	0.019526	0.019693
2^{12}	-	0.012939	0.010406	0.010588	0.010947	0.011261	0.011477
2^{14}	-	-	0.009506	0.008584	0.008625	0.008832	0.009005
2^{16}	-	-	-	0.008875	0.008520	0.008483	0.008550
2^{18}	-	-	-	-	0.008724	0.008617	0.008602
2^{20}	-	-	-	-	-	0.008711	0.008662
2^{22}	-	-	-	-	-	-	0.008703

Table 7.3: Coefficients of variation for least-cost paths on quadtree converted to uniform grid.

	Resolution of grid						
Cells in tree	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}	2^{22}
2^{10}	0.031752	0.025343	0.026472	0.020738	0.020696	0.022791	0.022793
2^{12}	-	0.021522	0.019204	0.017589	0.015322	0.015587	0.016655
2^{14}	-	-	0.019913	0.017236	0.015933	0.015417	0.015023
2^{16}	-	-	-	0.017846	0.016574	0.016289	0.016116
2^{18}	-	-	-	-	0.016962	0.016653	0.016586
2^{20}	-	-	-	-	-	0.016858	0.016734
2^{22}	-	-	-	-	-	-	0.016843

Table 7.4: Coefficients of variation for least-cost paths on septuatree converted to uniform grid.

	Resolution of grid						
Cells in tree	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}	2^{22}
2^{10}	0.024863	0.053702	0.068175	0.075478	0.079836	0.082583	0.083611
2^{12}	-	0.012939	0.034207	0.041270	0.045477	0.048126	0.049202
2^{14}	-	-	0.009506	0.015569	0.020066	0.022834	0.023825
2^{16}	-	-	-	0.008875	0.010925	0.012956	0.013814
2^{18}	-	-	-	-	0.008724	0.009286	0.009893
2^{20}	-	-	-	-	-	0.008711	0.008729
2^{22}	-	-	-	-	-	-	0.008703

Table 7.5: Coefficients of variation for least-cost paths on quadtree.

	Resolution of grid						
Cells in tree	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}	2^{22}
2^{10}	0.031752	0.057252	0.069741	0.064266	0.065157	0.068720	0.071806
2^{12}	-	0.021522	0.030693	0.041228	0.046264	0.031980	0.038793
2^{14}	-	-	0.019913	0.024454	0.023007	0.024641	0.026634
2^{16}	-	-	-	0.017846	0.018480	0.022130	0.017176
2^{18}	-	-	-	-	0.016962	0.015859	0.016558
2^{20}	-	-	-	-	-	0.016858	0.015210
2^{22}	-	-	-	-	-	-	0.016843

Table 7.6: Coefficients of variation for least-cost paths on septuatree.

7.3.1 Least-cost paths on non-uniform grids converted to uniform grids

When only using quadtrees and septuatrees as an intermediate storage format, we observe the results given in Tables 7.3 and 7.4. Similar to the results provided in section 4.4, we see that the coefficients of variation for septuatrees using hexagons are higher than for quadtrees using rectangles for all measurements performed.

Disregarding this, we can make an interesting observation. While we see the coefficient of variation increase by more than a factor of two when we decrease the number of cells in a quadtree from 2^{22} to 2^{10}, we can only observe a 35% increase for the septuatree. Still, on an absolute scale the septuatree still performs worse, so it is not yet possible to conclude that a septuatree has better characteristics with respect to preservation of the terrain.

7.3.2 Least-cost paths on non-uniform grids

When also using quadtrees and septuatrees as a structure to calculate the least-cost paths on directly, we see the results differ more radically, as shown in Tables 7.5 and 7.6. First of all, we must make the unfortunate observation that the use of the adaptive structures only adds an imprecision to the results of the computation. For each row in the table, the coefficients of variation increase as the resolution of the grid increases. This demonstrates that it is questionable whether computations of least-cost paths should be done on these structures directly.

Still, we do observe that for very high levels of sparseness in these trees, the least-cost paths computed on septuatrees start to become of better quality than their quadtree equivalents.

7.4 Open questions

In this chapter we have thoroughly described how primitive operations on septuatrees can be implemented, that can act as a basis for implementing larger algorithms that operate on terrains of various precision. Unfortunately, we have seen that the measurements on the quality of least-cost paths don't provide a practical use-case for this data structure yet. Still, these results don't limit the performance of septuatrees in general, meaning there are still many subjects in this area that can be researched:

- How do other conventional algorithms for grid terrains, such as flow accumulation and viewshed computation, behave when being executed on tree structures directly?
- Can the negative influence of using trees for least-cost path computation be cancelled out if the tree adheres to additional constraints (e.g. limiting the number of neighbouring grid cells)?
- In this chapter we only considered storing hexagons in the G-order. In which aspect do the described algorithms need to be modified to work on the curves and tilings presented in chapter 6?
- Can septuatrees be used to improve the running times algorithms that depend on efficient extraction of points inside a disc from a point cloud? For the tiling of the Gosper curve, it is known that its *Arrwwid number* is smaller than the for Z-order and the Hilbert curve.[6] Informally speaking, the Arrwwid number is the smallest number a such that any disk is covered by at most a relatively small tiles.

8 Conclusion

In this master's thesis we have attempted to determine the accuracy of algorithms for grid terrains when applied to alternative terrain models. Unfortunately we have seen that the results presented in each chapter make it hard to draw universal conclusions.

For both rotations on grids and flow accumulation using an angle-based weight function, we have seen that the use of hexagonal grid cells causes an improvement to the results computed. However, for both the least-cost paths and the slope-based weight functions for flow accumulation, we have observed the opposite, namely that regular grid cells perform the best.

It seems that this difference in performance can be attributed to the fact that every type of algorithm depends on a different property of a terrain model to work precisely. For the rotation we observed that interpolation on a smaller number of sample points causes an improvement, while for least-cost paths a greater freedom of directions to pass through the grid seems to be of importance.

This observation seems to suggest that for the questions mentioned in the introduction there is no single answer. Depending on the type of computation performed, a different terrain model may be preferred.

Looking at the chapters on space-filling curves and adaptive data structures for hexagonal grid cells, results do seem to be promising. The main contribution provided by this master's thesis in this area, is that it demonstrates that a recursive tiling of a grid does not have to be one of the prerequisites to provide efficient algorithms that operate on hierarchical adaptive grid structures. This has been demonstrated by the use of a septuatree.

Looking forward, the research performed in this master's thesis can be extended in multiple directions. These extensions can essentially be divided in two groups. On one hand, there are still many other algorithms one could use to compare different terrain models, including the computation of viewsheds. This research can both be focussed on practical analysis (e.g. performing measurements) and theoretical analysis (e.g. deriving bounds). On the other hand, the last two chapters of this thesis could potentially be used as a foundation for creating algorithms for new classes of space-filling curves and hierarchical adaptive grid structures.

Bibliography

[1] J. Akiyama et al. "Infinite Series of Generalized Gosper Space Filling Curves". In: *Proceedings of the 7th China-Japan conference on Discrete geometry, combinatorics and graph theory* (2007).

[2] A.L. Edmons. "Sommerville's missing tetrahedra". In: *Journal of Discrete and Computational Geometry* 37 (2 2007), pp. 287–296.

[3] R.A. Finkel and J.L. Bentley. "Quad trees a data structure for retrieval on composite keys". In: *Acta Informatica* 4 (1 1974), pp. 1–9.

[4] T.G. Freeman. "Calculating catchment area with divergent flow based on a regular grid". In: *Computers and Geosciences* 17 (3 1991), pp. 413–422.

[5] M. Gardner. *Penrose tiles to trapdoor ciphers and the return of Dr. Matrix*. Cambridge University Press, 1987.

[6] H.J. Haverkort. "Recursive tilings and space-filling curves with little fragmentation". 2010.

[7] H.J. Haverkort and J. Janssen. "Simple I/O-efficient flow accumulation on grid terrains". 2009.

[8] T. Hazel et al. "Terracost: Computing least-cost-path surfaces for massive grid terrains". In: *Journal of Experimental Algorithmics* 12 (2008), 1.9:1–1.9:31.

[9] D. Hilbert. "Über die stetige Abbildung einer Linie auf ein Flächenstück". In: *Mathematische Annalen* 38 (1891), pp. 459–460.

[10] M. Lanthier, A. Maheshwari, and J. Sack. "Approximating Weighted Shortest Paths on Polyhedral Surfaces". In: *Proceedings of the thirteenth annual symposium on Computational geometry* (1997), pp. 485–486.

[11] P. Linch. "The origins of computer weather prediction and climate modeling". In: *Journal of Computational Physics* (227 2008), pp. 3431–3443.

[12] T.K. Peucker et al. "The triangulated irregular network". In: *Proceedings of the International Symposium on Cartography and Computing: Applications in Health and Environment* 2 (1979), pp. 96–103.

[13] C. Qin et al. "An adaptive approach to selecting a flowpartition exponent for a multiple-flowdirection algorithm". In: *International Journal of Geographical Information Science* 21 (4 2007), pp. 443–458.

[14] P. Quinn et al. “The prediction of hillslope flow paths for distributed hydrological modelling using digital terrain models”. In: *Hydrological Processes* 5 (1 1991), pp. 59–79.

[15] L. Raisanen and R.M. Whitaker. “Comparison and evaluation of multiple objective genetic algorithms for the antenna placement problem”. In: *Journal of Mobile Networks and Applications* 10 (1-2 2005), pp. 79–88.

[16] D.M.Y. Sommerville. “Space-filling Tetrahedra in Euclidean Space”. In: *Proceedings of the Edinburgh Mathematical Society* (1923), pp. 49–57.

[17] L. Toma et al. “Flow computation on massive grids”. In: *Proceedings of the 9th ACM international symposium on Advances in geographic information systems* (2001), pp. 82–87.