# Generation of Zoomable Maps with Rivers and Fjords

## Extended 2025 version

Torben Æ. Mogensen and Emil N. Isenbecker

[1] Department of Computer Science, University of Copenhagen
torbenm@di.ku.dk
[2] Department of Computer Science, University of Copenhagen
lwr500@alumni.ku.dk

**Abstract.** This paper presents a method for generating maps with rivers and fjords. The method is based on recursive subdivision of triangles and allows unlimited zoom on details without requiring generation of a full map at high resolution.

## 1 Introduction

Procedurally generated maps are often used in computer games, where they allow variability when playing a game multiple times. For strategy games, a full map of the entire playing area is typically generated, but for first-person-view games, storing a full map with all details can be unrealistic, so details are typically generated on the fly. This requires *zoomable* maps, where a high-resolution local map can be generated without having to generate the full map at this resolution. You may even want to generate different levels of detail for the near and far parts of the map. A similar requirement exists for pen-and-paper role-playing games, where you might want both a world map and local maps for individual islands and continents.

Many algorithms exist for generating topographical maps, basically generating altitude information for every point and then colouring the points according to some colour map. Examples include the diamond-square algorithm [4], the square-square algorithm [7], Fourier-transform-based methods [1], great circle displacement [2, 9, 12], simulated plate tectonics [3], and tetrahedral subdivision [8]. Most of these suffer from lack of zoomability: Generating a high resolution image of a small section of a map requires almost as much computation time as generating a high-resolution image of the full map. Exceptions are methods based on recursive subdivision such as diamond-square and tetrahedral subdivision.

None of these methods natually support generation of rivers. For any fixed-size map, rivers can be placed after the map is generated, but this prevents zooming, as rivers can not be consistently placed on a local map without knowing their placement on the global map – we need, at least, to know where rivers flow into the selected map fragment. So the only zoom option is to pregenerate the

full map at the resolution of the highest zoom and then show only the selected part. But that is very costly. Ideally, a zoomed map should be generated at a cost which is close to proportional to the size (in pixels) of the shown portion of the map and not of the full map.

Generation of realistic river networks using Strahler-Horton analysis has been proposed [11], but these do not follow any terrain. The challenge is to co-generate terrain and rivers while being zoomable.

This paper shows a method for doing just that. The method is based on triangle subdivision, so it suffers from the same kind of artefacts as the diamond-square algorithm, but it does generate zoomable maps with decent-looking river networks and fjords, which to the authors' knowledge has not been done previously. Figure 1 shows an example of such a map at four different degrees of zoom. On the top-left picture it is clear that the river network actually continues as trenches in the fjord.

## 2 Map Generation by Triangle Subdivision

The first step is to present a simple map generation method using triangular subdivision. We use isosceles, right-angled triangles as shown in Figure 2(a). We divide this into two half-size triangle by drawing a new edge $e_3$ from the vertex $V_0$ to the middle of the opposite edge $e_0$, splitting this into two new edges $e_4$ and $e_5$, as shown in Figure 2(b). The new vertex at the middle of $e_0$ is called $V_3$.

Each vertex is represented by a record that contains the coordinates $(x, y)$ of the vertex (each ranging from 0.0 to 1.0), the altitude $h$ at the vertex (ranging from -1.0 to 1.0), and a pseudorandom value $s$ in the range -1.0 to 1.0. We generate the attributes of $V_3$ by the formulae:

$$\begin{aligned} V_3.x &= (V_1.x + V_2.x)/2 \\ V_3.y &= (V_1.y + V_2.y)/2 \\ V_3.s &= \mu(V_1.s,\, V_2.s) \\ \delta \quad &= k_1|V_1 - V_2| + k_2|V_1.h - V_2.h| \\ V_3.h &= (V_1.h + V_2.h)/2 + \delta V_3.s \end{aligned}$$

where $\mu$ is a mixing function that takes two pseudorandom values and produces a new pseudorandom value. It must be the case that $\mu(s_1, s_2) = \mu(s_2, s_1)$ for all $s_1$ and $s_2$. We will also use $\nu(s) = \mu(s, s)$.

$\delta$ is a measure of how much the altitude can differ from the average of $V_1.h$ and $V_2.h$. $k_1$ and $k_2$ are constants that determine how much the altitude varies as a function of the distance between the vertices ($|V_1 - V_2|$) and the altitude difference between the vertices ($|V_1.h - V_2.h|$), respectively. For the examples used in this paper, we use $k_1 = 0.32$ and $k_2 = 0.55$. It is possible (though unlikely) that this gives a $V_3.h$ outside the range -1.0 to 1.0. If this happens, we cap it to be inside the range.

Note that $V_0.h$ is not used when finding $V_3.h$. This is because the edge $e_0$ can be shared with another triangle that does not have access to $V_0$. This is the main difference between this method and the diamond-square method: Only

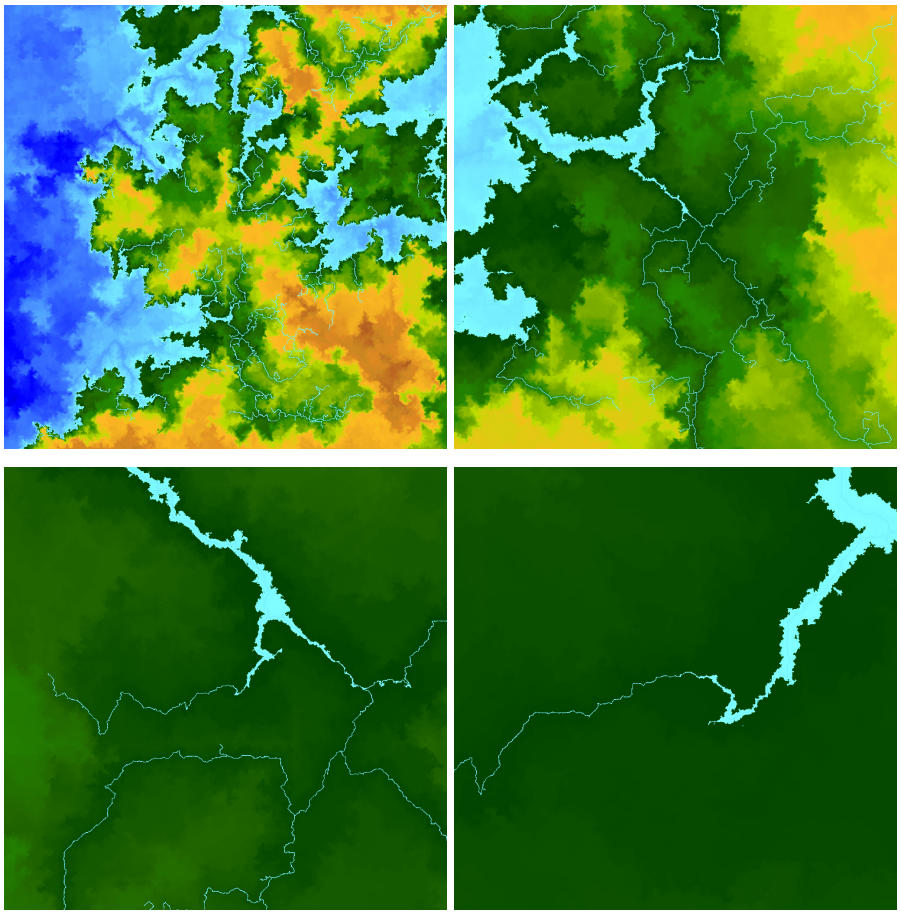
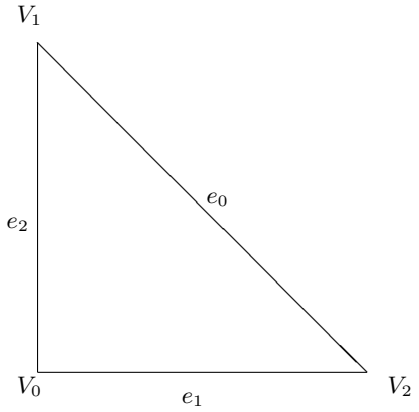
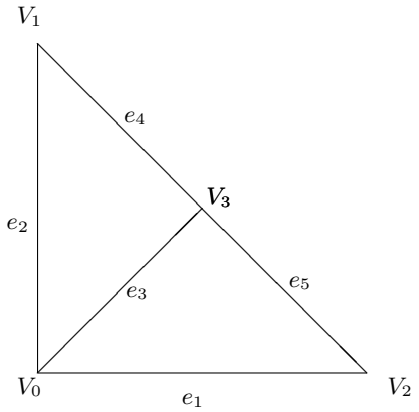


**Fig. 1.** Map showing rivers and fjords at zoom 1, 5, 25 and 125




(a) Isosceles, right-angled triangle.




(b) Subdivision of triangle.

**Fig. 2.** Triangle Subdivision

two altitudes are averaged when finding the altitude of a new point, where the diamond-square method averages four altitudes when finding the altitude of a new point.

We continue recursively subdividing the triangle into ever-smaller triangles, until a triangle is the size of one pixel. At this point, we use $V_3.h$ to colour the corresponding point on the map. The maps shown in this paper uses blue shades for negative altitudes and green, brown, and white shades for positive altitudes. A map produced by this simple algorithm is shown in Figure 3.

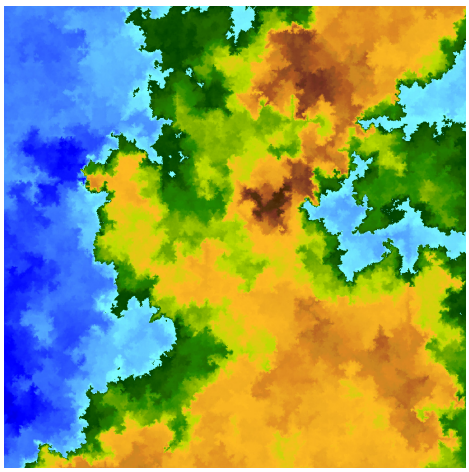


**Fig. 3.** Map generated by triangle subdivision

## 3 Adding Rivers

We will now extend the simple map generator to include rivers. We will do so using the edges of the triangle: An edge $e_i$ has an *optional* altitude attribute $e_i.h$. If this attribute is present, it indicates that *somewhere* on edge $e_i$ a river crosses at altitude $e_i.h$. If the attribute is absent, no river flows through the edge. When subdividing a triangle, we use the attributes on $e_0$, $e_1$ and $e_2$ together with the attributes on $V_0$, $V_1$ and $V_2$ to generate the attributes for $e_3$, $e_4$, and $e_5$. Moreover, we allow the altitude attribute $e_0.h$ to influence $V_3.h$. Basically, we want that, when the subdivision finishes, $V_3.h$ denotes the altitude of the corresponding point of the river (if any).

### 3.1 The long edge

We recall that the attributes of $V_3$, $e_4$ and $e_5$ can only depend on the attributes of $V_1$, $V_2$ and $e_0$, since the edge $e_0$ can be shared with another triangle that does

not have access to $V_0$, $e_1$ and $e_2$. So let us start with $V_3$, $e_4$ and $e_5$ and then handle $e_3$ later.

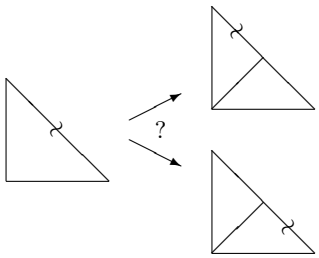



**Fig. 4.** When there is a river on the long edge

We noted that $e_0$ has an attribute $e_0.h$ if and only if a river crosses $e_0$. If this is the case, we have to choose whether this river crosses the half-edge $e_4$ or the half-edge $e_5$, as shown in Figure 4. We do not allow a river to cross both $e_4$ and $e_5$, as that could potentially lead to circular river segments that do not connect to sea. We use a simple method for this: If $e_0.h$ is closer to $V_1.h$ than to $V_2.h$, the river flows through $e_4$, otherwise it flows through $e_5$. The attribute of $e_0$ is copied to the chosen half-edge, and the other half-edge has no attribute. Ties can be broken by looking at the other attributes of $V_1$ and $V_2$, but ties are extremely rare when floating-point numbers are used. If a river flows through $e_4$, $e_4.h$ is used instead of $V_1.h$ when determining $V_3.h$, and if a river flows through $e_5$, $e_4.h$ is used instead of $V2.h$. If no river flows through $e_0$, we determine $V_3.h$ as before. We can describe the determination of $e_4$, $e_5$, and $V_3.h$ with the following pseudocode:

```
if e_0 has an attribute e_0.h, then
   if |e_0.h − V_1.h| < |e_0.h − V_2.h| then
      e_4.h = e_0.h;
      e_5.h = Absent;
      V_3.h = (e_4.h + V_2.h)/2 + δV_3.s
   else
      e_5.h = e_0.h;
      e_4.h = Absent;
      V_3.h = (e_5.h + V_1.h)/2 + δV_3.s
else
   e_5.h = Absent;
   e_4.h = Absent;
   V_3.h = (V_1.h + V_2.h)/2 + δV_3.s
```

The other attributes of $V_3$ are calculated as before.

### 3.2 The short edge $e_3$

The complicated part is determining the attribute (if any) of $e_3$. Note that this can (and does) depend on all vertices and all edges.

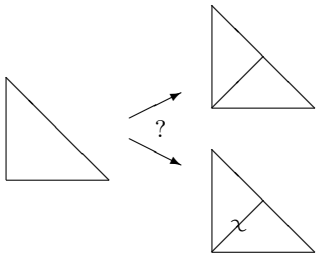



**Fig. 5.** When there is no river on any outer edge

**No surrounding edge has a river** We start with the case where there is no river on $e_1$, $e_2$, $e_4$ or $e_5$. We then need to determine if a river should flow through $e_3$ or not, as illustrated in Figure 5. If $V_1$ is above water and $V_2$ is below water, there is a possibility of a river flowing from somewhere in the triangle $V_1$, $V_0$, $V_3$ to somewhere in the triangle $V_2$, $V_0$, $V_3$, but only if there is no closer exit. So we rule that both $V_0$ and $V_3$ should be higher than $V_2$. We also prefer rivers with some steepness over very flat rivers, so we want $V_1$ to be higher above water than $V_2$ if they are far apart than if they are close. Additionally, we want the height of the river at $e_3$ to be between $V_2.h$ and the minimum of $V_0.h$ and $V_3.h$. In pseudocode, this is

```
if V1.h > k3 ∧ V2.h < k4 ∧ V2.h < V0.h ∧ V2.h < V3.h
then
   e3.h = β(V2.h, min(V0.h, V3.h), μ(V0.s, V3.s));
else
   e3.h = Absent;
```

Where $k_3$ and $k_4$ are the minimal above-land and below-sea altitudes for $V_1.h$ and $V_2.h$. In the examples in this paper, we have use $k_3 = 0.1$ and $k_4 = -0.1$. $\beta(h_1, h_2, s)$ finds an altitude between $h_1$ and $h_2$ using the pseudorandom value $s$. This should tend towards the middle of the interval, so (since $-1 \leq s \leq 1$), we define $\beta(x, y, s) = (x + y + s^3(x - y))/2$. Other definitions can be used as long as the result tends towards the average of $x$ and $y$.

We have in the examples shown here chosen that there is always a river if the preconditions are met, but you can add a pseudorandom element if you want fewer rivers.

The case where $V_1$ is below water and $V_2$ is above water is handled symmetrically.

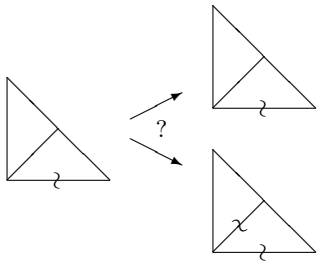



**Fig. 6.** When there is a river on one outer edge

**One surrounding edge has a river** If only one of the edges $e_1$, $e_2$, $e_4$ and $e_5$ is crossed by a river, we want to see if we should extend this through $e_3$. We show the case if a river passing through $e_1$ (s shown in Figure 6, as the other cases are symmetric.

If there is a river through $e_1$, $V_1.h < 0$, and $V_1.h < e_1.h$, the river might flow down towards $V_1$, but only if there is not a shorter path to the sea near $V_3$. So if $V_1.h < 0$, $V_1.h < e_1.h$ and $V_3.h > 0$, we add a river to $e_3$ with altitude between that of $e_1$ and $V_1$, so, $e_3.h = \beta(e_1.h,\, V_1.h,\, \mu(V_3.s, V_0.s))$.

If, on the other hand, $V_1.h > e_1.h$, there is a possibility that the river flows from somewhere in the triangle $V_1$, $V_0$, $V_3$ towards $e_1$, but only if all three vertices are higher up than $e_3$. Also, we might not want to always extend the river all the way to the top of a mountain, so we add a chance that it will not extend across $e_3$.

I pseudocode, we handle the case of a river only through $e_1$ by

```
if V1.h < 0 ∧ V1.h < e1.h ∧ V3.h > 0 then
   e3.h = β(V1.h, e1.h, μ(V0.s, V3.s));
else if V1.h > e1.h ∧ V0.h > e1.h ∧ V3.h > e1.h
   then if |μ(V0.s, V3.s)| < k5 then
      e3.h = β(e1.h, min(V1.h, V0.h, V3.h, ν(V1.s));
   else
      e3.h = Absent;
else
   e3.h = Absent;
```

Where $k_5$ is the probability that we extend a river upwards if the preconditions are met. In the examples used in this paper, $k_5 = 0.7$.

There are symmetric cases for rivers going through $e_2$, $e_4$, or $e_5$. Where we use $V_3$, above, we use $V_0$ when extending rivers through $e_4$ or $e_5$. I.e., we do not use a vertex on the same edge as the river.

**Two surrounding edges have rivers** If the two edges that have rivers are on opposite sides of $e_3$, the case is simple: There *will* be a river on $e_3$ and its altitude is between that of the two other river edges. There are three such cases:

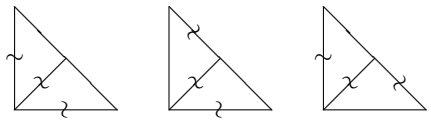


**Fig. 7.** When there are rivers on two opposing outer edges

rivers on $e_1$ and $e_2$, rivers on $e_1$ and $e_4$, and rivers on $e_2$ and $e_5$ (since there can not be rivers on both $e_4$ and $e_5$). We illustrate these cases in Figure 7.

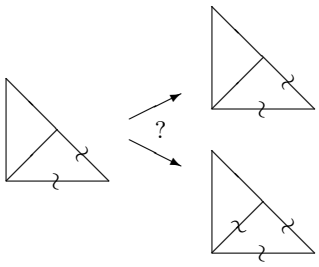



**Fig. 8.** Potentially branching river

The case where the two river-carrying edges are on the same side of $e_3$ is more interesting. It is clear that a river flows from one edge to the other without crossing $e_3$, but there is a possibility that the river branches with a branch entering through $e_3$, as illustrated in figure 8. This branch must be higher up than the lowest of the river-carrying outside edges, as we do not allow a river to split downwards.[3] We want all three vertices of the triangle on the opposite edge of $e_3$ to be higher than the lowest of the river-carrying outside edges, and the river on $e_4$ must have an altitude between the lowest of the three vertices and the lowest of the river-carrying outside edges. Additionally, we might not always want a branch, so we add a probability.

The case where the edges $e_1$ and $e_5$ carry rivers is shown in pseudocode below. The case where the edges $e_1$ and $e_5$ carry rivers is symmetric.

```
if min(V1.h, V0.h, V3.h) > min(e1.h, e5.h)
then if |μ(V0.s, V3.s)| < k6|V1 − V2| then
        e3.h = β(min(V1.h, V0.h, V3.h), min(e1.h, e5.h), ν(V1.s))
     else
        e3.h = Absent;
else
     e3.h = Absent;
```

[3] This can happen in river deltas, but we do not handle this case here.

where $k_6|V_1 - V_2|$ is the probability that the river branches if the preconditions are met. Note that this means that, on a larger scale, rivers are more likely to branch than on a smaller scale. We use $k_6 = 2$.

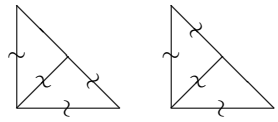


**Fig. 9.** When there are rivers on three outer edges, they must connect

**Three surrounding edges have rivers** This case is also relatively simple: There is always a river crossing $e_3$, as illustrated in Figure 9. The only nontrivial part is finding the altitude of this river. The two higher river edges will flow towards the lowest river edge, so the river on $e_3$ must be higher than the lowest river edge but lower than the river(s) on the opposite side of the lowest river.

The case where there are rivers on $e_1$, $e_2$, and $e_4$ is handled by the pseudocode

$$e_3.h = \beta(e_1.h,\, \min(e_2.h, e_4.h))$$

The case where there are rivers on $e_1$, $e_2$, and $e_5$ is handled symmetrically.

There are no cases with rivers on both $e_4$ and $e_5$, so we have handled all possible cases.

Figure 1 shows a map generated by these rules. Compare the top-left image to Figure 3, which is the same map but without rivers. Note how the rivers "carve" fjords and river valleys into the land. Fjords are generated when the river has an altitude that is below sea level and "pulls" nearby land below sea as well.

## 4 Variant: Islands in Fjords

If we allow rivers on both halves ($e_4$ and $e_5$) of a split long edge ($e_0$) of the isosceles triangle, it can potentially lead to disconnected circular rivers. But if we do do only if $e_0.h < k_7$, for some negative $k_7$ (the examples use $k_7 = -0.1$), such circularity can only happen in fjords. So instead of circular rivers, we get islands in fjords and occasional narrow straits crossing land. Using this variant, we get the maps shown in Figure 10. The islands are in the fjord near the top of the map.

We have to modify splitting of $e_0$ to

```
if e_0 has an attribute e_0.h, then
   if e_0.h < k_7 and |ν(e_0.s)| < k_8 then
      e_4.h = e_0.h;
      e_5.h = e_0.h;
      V_3.h = (e_4.h + e_5.h + V_2.h)/3 + δV_3.s
   else if |e_0.h − V_1.h| < |e_0.h − V_2.h| then
      e_4.h = e_0.h;
      e_5.h = Absent;
      V_3.h = (e_4.h + V_2.h)/2 + δV_3.s
   else
      e_5.h = e_0.h;
      e_4.h = Absent;
      V_3.h = (e_5.h + V_1.h)/2 + δV_3.s
else
   e_5.h = Absent;
   e_4.h = Absent;
   V_3.h = (V_1.h + V_2.h)/2 + δV_3.s
```

where $k_8$ is a probability of copying the river. We have used $k_8 = 0.15$. It is not entirely realistic that $e_4.h = e_5.h$, but if we change one or the other, we can get rivers flowing upward.

We also need to add cases where there are rivers on both $e_4$ and $e_5$. If no other edges or all other edges have rivers, we do not add a river on $e_3$. If one other edge ($e_1$ or $e_2$) has a river, we add a river on $e_3$ with an altitude between $e_1.h$ and $e_4.h$ or between $e_2.h$ and $e_5.h$, depending on whether $e_1$ or $e_2$ has a river.

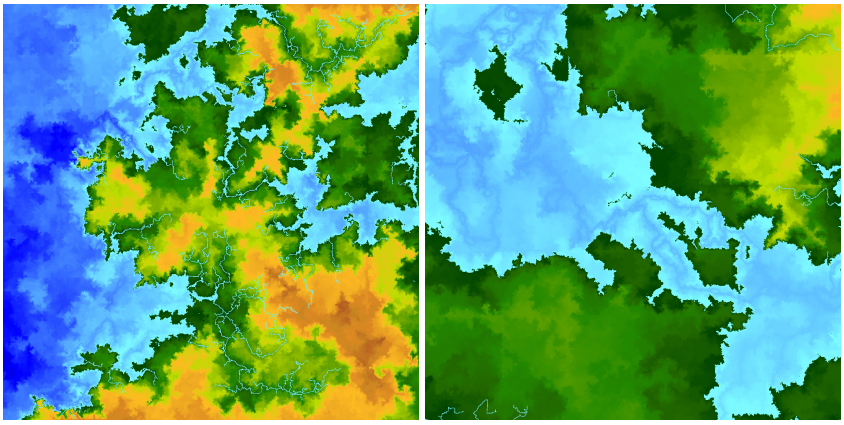


**Fig. 10.** Islands in fjords

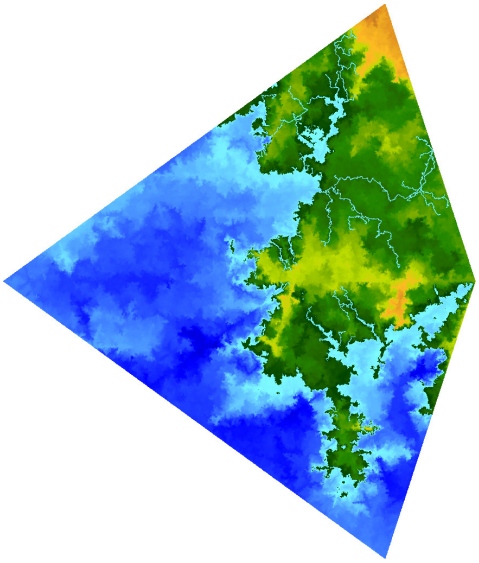


**Fig. 11.** Penrose-tiled map

## 5 Variant: Penrose Tiles

To reduce the axis-aligned artefacts, we can use Penrose tiles [5,6,10], which form aperiodic tilings. We use the P2-tiling that uses "kites and darts", both formed by joining two Robinson triangles: isoscleles triangles where the ratio of the long and short sides is the golden ratio $\phi = \frac{\sqrt{5}+1}{2}$. These can be subdivided into smaller Robinson triangles as shown in Figure 12. The divided edge is split at a ratio of $\phi$ to 1. Since the split is uneven, mirroring matters, so we have indicated the orientation of triangles with arrows. Two of the top-type (acute) triangles form a kite shape, and a map in this shape created by subdividing these is shown in Figure 11. We have used the same rules for river placement as for the right-angles triangles, but the altitude of a new vertex on a split edge without rivers use a weighted average of the endpoints (with random displacement), so the altitude of the new point is (statistically) closer to the nearest of its neighbours. We use

$$V_3.h = (\phi - 1)V_1.h + (2 - \phi)V_2.h + \delta V_3.s$$

where $V_1$ is the vertex closest to the cut point.

## 6 Variant: Scalene Triangles

We can generalise the method to use scalene triangles, where both the half-square and Robinson triangles are isosceles triangles.

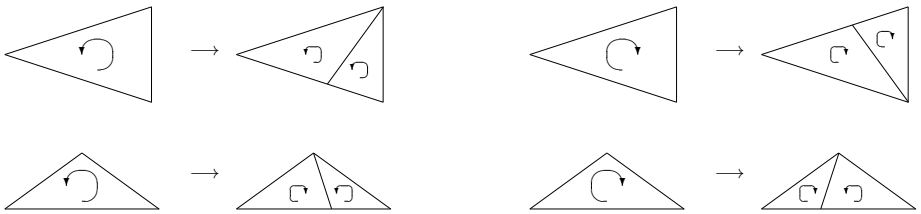
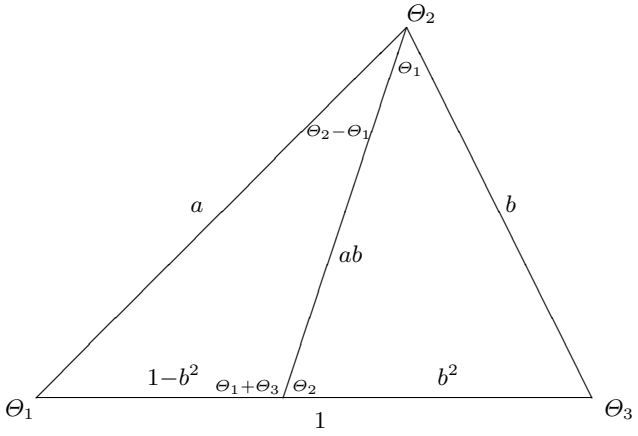


**Fig. 12.** Subdividing Robinson triangles



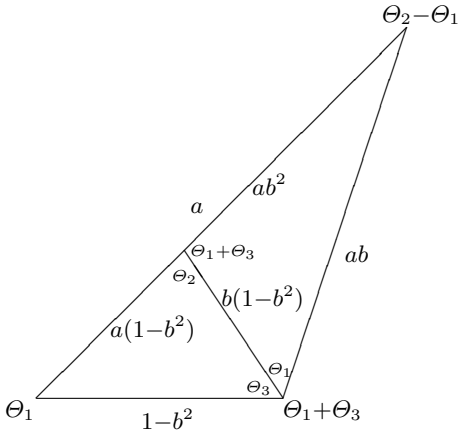




**Fig. 13.** Subdividing scalene triangles

We start with a triangle with side lengths $1 > a > b$. This can be divided by cutting the long edge at proportions $1 - b^2$ and $b^2$, as shown in Fig. 13, top. $\Theta_a$, $\Theta_2$, and $\Theta_3$ are the angles of the corners.

The rightmost triangle has dimensions $b$, $ab$, $b^2$, which is congruent (though mirrored and rotated) with the original triangle with scaling factor $b$. The leftmost triangle has dimensions $a$, $1-b^2$, $ab$.

Fig. 13, bottom, shows how this triangle can be subdivided. Note that the bottom-left triangle is congruent with the initial triangle with scaling factor $1-b^2$ and that the top-right triangle is congruent (mirrored and rotated) with the second triangle with scaling factor $b$. Note, also, that the ratio with which we divide the longest edge is the same: $1-b^2$ to $b^2$. Since the three edges are of different lengths, we do not need to keep track of orientation, as we did for Roninson triangles.

Using scalene triangles allows a spherical map to be approximated by, for example, a disdyakis dodecahedron, a 48-faced polyhedron. A triangular map with the dimensions needed to construct a disdyakis dodecahedron is shown in Fig. 14. The edge lengths of this triangle are 1, $a = \frac{3\sqrt{12+6\sqrt{2}}}{2\sqrt{60+6\sqrt{2}}}$, and $b = \frac{\sqrt{30-3\sqrt{2}}}{2\sqrt{60+6\sqrt{2}}}$.

When combining triangles to form, say, polyhedra, triangles meeting at a vertex must have the same seed and altitude values at the meeting corners and when triangles share an edge, the river attribute on this edge must be the same for both triangles.

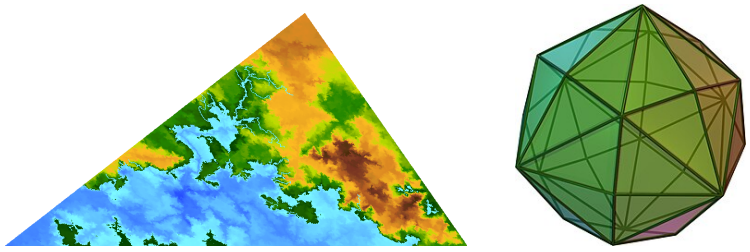


**Fig. 14.** Triangle suited for constructing a disdyakis dodecahedron
Image of disdyakis dodecahedron courtesy of Wikipedia

## 7 Variant: Wide Rivers

In the above, rivers are one pixel wide, no matter how much you zoom (the wider waterways in Figure 1 are fjords at sea level). We can assign widths as well as altitudes to rivers at edges. When a river is extended (Figure 6), and the opposing vertex is higher, the new river edge has a smaller width, and when

the opposing vertex is lower, it has a greater width. When a branch is made (Figure 8), it gets a width that is smaller than the main river.

What makes the rivers wide on the generated maps is that in the cases shown in Figures 6 and 8 we now always add a river on the new edge if the rivers on the old edges are wider than the size of the triangle (in the one-river case, half as wide to avoid very pointy river bends). The new river is as wide as the main river. A river pixel is generated if just one edge of a pixel-wide triangle has a river edge, which is the case for most of the triangles created by further subdivision.

Figure 15 compares a zoomed-in map section with pixel-wide rivers and wider rivers. Note that the river width is not constant – the rivers that branch off the main river are narrower than the main river and narrows down further towards their origins.

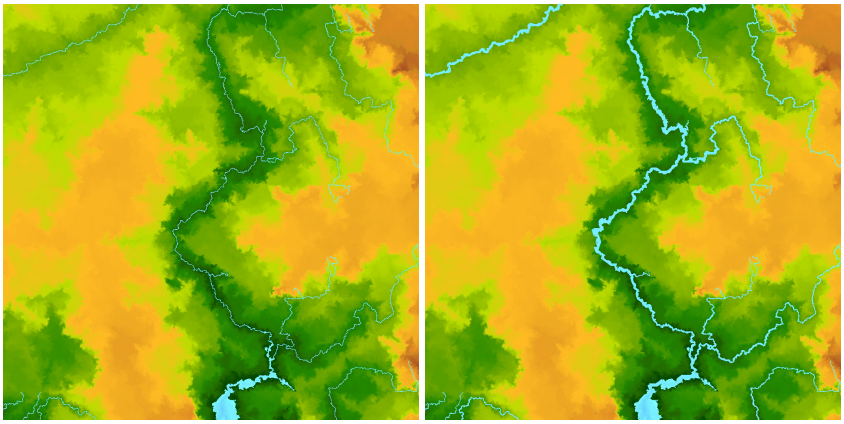


**Fig. 15.** Narrow and wide rivers

## 8 Assessment

The method generates zoomable maps with rivers that look somewhat realistic, but they suffer from the same problems that maps generated by recursive subdivision usually do: There are some artefacts due to the chosen grid, and the rivers do not always flow naturally: They do not always take the steepest route down, and sometimes there are no rivers where you would expect rivers to be. Only altitudes and rivers are generated, other details such as vegetation and climate are not handled.

Nevertheless, we believe the maps generated by the method can be used for gaming – both computer and table-top games – possibly with extensions for creating vegetation and climate details.

The values for the constants $k_1, \ldots k_6$ are chosen somewhat arbitrarily by looking at generated maps and see what "looks good". There are possibilities for

varying these and even adding more "magic" constants to modify the statistics of the produced maps.

The maps shown in the paper are all made by a straight-forward implementation of the algorithm in F# running on Mono, and all use $1023 \times 1023$ pixels.

Relatively few edges have rivers, so even though handling rivers is fairly complex, this is done relatively rarely, so rivers add almost nothing to the running time. The time to generate unzoomed maps with or without rivers is about 1.4 seconds. The $125\times$ zoomed map takes about 1.5 seconds, so the time is also almost independent on the zoom factor. All executions are done using single-threaded F# compiled to .NET and executed using Mono on an Intel Core i7-10510U processor running at 1.80GHz. No attempts have been made at optimising the code.

Using integers instead of floating points for coordinates, seeeds, and altitudes can speed calculation up, but the reference implementation uses floats to be close to the description in this paper. It is also fairly easy to exploit multiple cores: The first few subdivisions can make one of the recursive calls run on a new core. It is also possible to vectorise the method: Each pixel is generated individually. Starting with the original triangle, it is determined in which half the pixel is located, and subdivision is repeated on this triangle only, and so on until the triangle is smaller than the pixel. Each such loop is running on a single core, but different cores can process different pixels in parallel.

The Penrose-tile and scale-triangle variants takes a bit longer to generate than when subdividing right-angled isosceles triangles, and can not (easily) use integer coordinates. But they generate fewer grid-aligned artefacts.

## 9 Future Work

Future work may include extending the method to tetrahedral subdivision [8]. This generates a planet map by projecting each map point onto a sphere contained in an irregular tetrahedron, and then recursively subdividing this while discarding tetrahedra that do not contain the point. Figure 16 illustrates tetrahedral subdivision.

While tetrahedral subdivision resembles triangle subdivision, adding rivers is not trivial: With triangles, rivers can float from edges to edges, as all edges will be part of the final map. With tetrahedra, an edge may or may not intersect the surface of the sphere (and may do so twice). It may work to say that a river can only be generated on an edge that intersects the sphere exactly one, and when subdividing the edge, the river is put on the half edge that intersects the sphere, but that is unlikely to give natural-looking river flows. Additionaly, an edge can be shared by more than two tetrahedra, so placing a river there is likely to give three-way or four-way river joins. It may be better to place rivers on faces instead of edges, as a face is shared only by two tetrahedra. Again, we can generate a river on a face only if it intersects the sphere, and when subdividing a face (which happens when a tetrahedron is cut in two), the river can be placed on a sub-face that intersects the sphere. There may be a choice, which can be

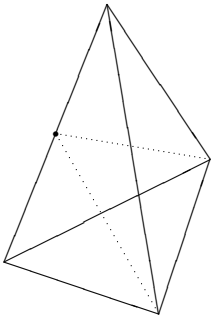


**Fig. 16.** Cutting a tetrahedron in two

resolved similar to how it is done with triangle subdivision. A problem is that rivers on faces can not affect altitudes on the map, as the altitude on the new point can only depend on attributes of the edge on which it is positioned and the end points of this edge. So it may be impossible to make rivers flow downwards only.

## References


1. Poul Bourke. Frequency synthesis of landscapes (and clouds). `http://paulbourke.net/fractals/noise`, 1997.
2. Poul Bourke. Modelling fake planets. `http://paulbourke.net/fractals/noise`, 2000.
3. C. Burke. Plate tectonics. `https://web.archive.org/web/20091026160900/http://www.geocities.com/Area51/6902/t_plate.html`, 1996.
4. A. Fournier, D. Fussel, and L. Carpenter. Computer rendering of stochastic models. *Communications of the ACM*, 25(6):371–384, June 1982.
5. Martin Gardner. Extraordinary nonperiodic tiling that enriches the theory of tiles. *Scientific American*, 236(1):110–121, 1977.
6. Martin Gardner. *Penrose tiles to trapdoor ciphers*. Recreational Mathematics. W. H. Freeman and Company, New York, 1989.
7. Gavin S. P. Miller. The definition and rendering of terrain maps. In *SIGGRAPH 1986 Conference Proceedings (Computer Graphics, Volume 20, Number 4, August 1986)*, pages 121–135, 1993.
8. Torben Ægidius Mogensen. Planet map generation by tetrahedral subdivision. In *Proceedings of the 7th International Andrei Ershov Memorial Conference on Perspectives of Systems Informatics*, PSI'09, pages 306–318, Berlin, Heidelberg, 2010. Springer-Verlag.
9. John Olsson. Fractal worldmap generator. `http://www.lysator.liu.se/~johol/fwmg/fwmg.html`, 2004.
10. Roger Penrose. Pentaplexity a class of non-periodic tilings of the plane. *The mathematical intelligencer*, 2(1):32–37, 1979.
11. Xavier Gérard Viennot. Trees everywhere. In *Proceedings of the 15th Colloquium on Trees in Algebra and Programming*, CAAP '90, pages 18–41, Berlin, Heidelberg, 1990. Springer-Verlag.

12. R. P. Voss. Random fractal forgeries. *Fundamental Algorithms for Computer Graphics*, 17:805–835, 1985.