Vibration of the Koch drum

A preprint version of a "Mathematical graphics" column from

Mathematica in Education and Research.

Mark McClure

mcmcclur@unca.edu

Department of Mathematics
University of North Carolina at Asheville
Asheville, NC 28804

Abstract

The Koch snowflake is a well known self-similar set whose boundary is a fractal curve. Suppose a vibrating membrane with fixed boundary, i.e. a drum, is shaped like a Koch snowflake. The fundamental modes of vibration of this drum can be modelled by the eigenfunctions of the Laplacian on the snowflake. These, in turn, can be approximated by a discrete problem leading to a matrix eigenvalue problem.

■ Mathematica Initializations

1. Introduction

■ 1.1 The problem

Readers of this column are surely familiar with the two dimensional self-similar set with fractal boundary called the Koch snowflake. Figure 1 illustrates the decomposition of this set into seven copies of itself - six scaled by the factor $1/3$ and one scaled by the factor $1/\sqrt{3}$.

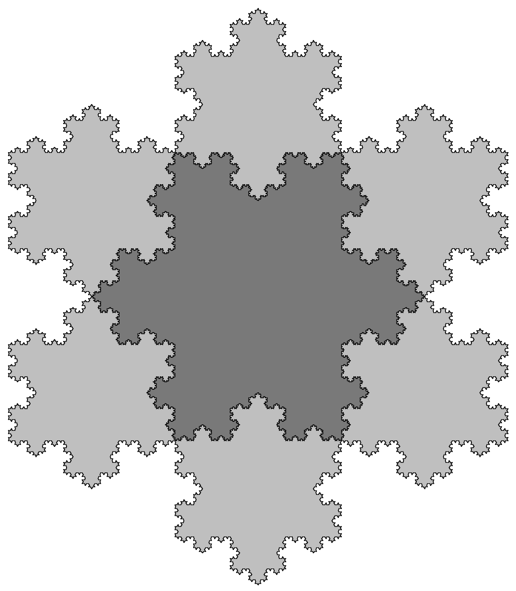


Figure 1: The Koch snowflake

Suppose a vibrating membrane with fixed boundary has the shape of a Koch snowflake. What types of vibrational modes are possible? This natural question goes back at least to Berry's work in 1979 [1] and many papers studying the problem have appeared over the years. In the mathematical formulation of the problem, the natural modes of vibrations are described by the eigenfunctions of the Laplacian on the region. Most studies approximate the Koch drum with a discrete grid of points and then translate the problem to a matrix eigenvalue problem. This is the approach taken here using the grid of points recently proposed in [2].

■ 1.2 The plan of attack

The basic wave equation is $w_{tt} = c^2 \Delta w$, where $w(x, t)$ is a function of time t and of the spatial variable x restricted to lie in some specified domain. Assuming that $w(x, t) = u(x)\, g(t)$ and separating variables, we obtain the equations $g'' = -\lambda \mathrm{c}^2\, g$ and $\Delta u = -\lambda\, u$. Note that any solution of the equation for u is simply an eigenfunction of the Laplacian on the domain in question and any such u defines a possible shape of the vibration called a fundamental mode. The possible values of λ dictate the frequencies of the fundamental modes.

For example, if we model the vibration of a string by restricting x to lie in the unit interval, then the equation for u is simply $u'' = -\lambda\, u$. We may specify the endpoints as fixed by insisting that $u(0) = u(1) = 0$. The eigenvalues for this boundary value problem are $\lambda = (n\, \pi)^2$ and the corresponding eigenfunctions are $u(x) = \sin(n\, \pi\, x)$. The leads to the familiar sinusoidal type modes of vibration for a string.

For us, of course, x is a two-dimensional variable permitted to range throughout the Koch snowflake and we will assume that $w(x, t) = 0$ for all x on the boundary of the snowflake. There is no simple analytic formula for the solution and the problem is not amenable to the algorithms incorporated in **`NDSolve`**. Thus, we will need to develop our own numerical technique using finite differences based on the approximating grid shown in figure 2. (The different shades of the vertices will be explained shortly.)

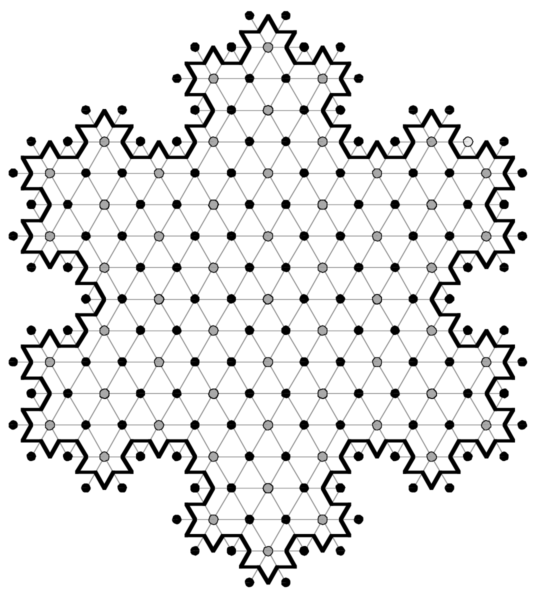


Figure 2: The approximating grid

Let x be an interior point of this grid - i.e. an element of the grid and inside the snowflake. Let N denote the set of six nearest neighbors at the distance h from x, possibly including some points outside of the snowflake. Then the Laplacian of a function u at x can be approximated using a second order difference quotient:

$$\Delta\, u(x) \approx \frac{2}{3\,h^2}\left(\left(\sum_{y\in N} u(y)\right) - 6\,u(x)\right).$$

Note that formula 0 states that the Laplacian of u at an interior grid point can be approximated with a linear combination of nearby values of u. We would like to eliminate reference to the exterior grid points, called ghost points in [2], and we must enforce the boundary condition. If we view this grid as a graph, then each ghost point is adjacent to precisely one interior grid point. (In this interpretation, however, multiple ghost points can occur at the same location. For example, there are three ghost points at the light vertex near the upper right of figure 2 and each of these are adjacent to a different point in the interior grid.) Now let x be a ghost point and let x' be the interior point adjacent to x. Symmetry properties of the Laplacian imply that boundary conditions can be enforced by setting $u(x) = -u(x')$.

Now since the value at a ghost point is related to the value at its interior neighbor, it is easy to remove reference to the ghost points. To do so, let x be an interior grid point, let N' denote its set of nearest interior neighbors (not including ghost points), and let $n' = \#(N')$. Then, since each ghost point contributes $-u(x)$ to the sum in formula 0, the approximation may be rewritten

$$\Delta\, u(x) \approx \frac{2}{3\,h^2}\left(\left(\sum_{y\in N'} u(y)\right) - (12 - n')\,u(x)\right).$$

This again expresses $\Delta\ u(x)$ is a linear combination of nearby values of u and uses only interior grid points. Thus formula 0 may be described as multiplication of the vector of values of u at the interior grid points by the appropriate matrix. The eigenvalues of this matrix are approximations to the eigenvalues of the Laplacian and the eigenvectors may be used to approximate the eigenfunctions of the Laplacian.

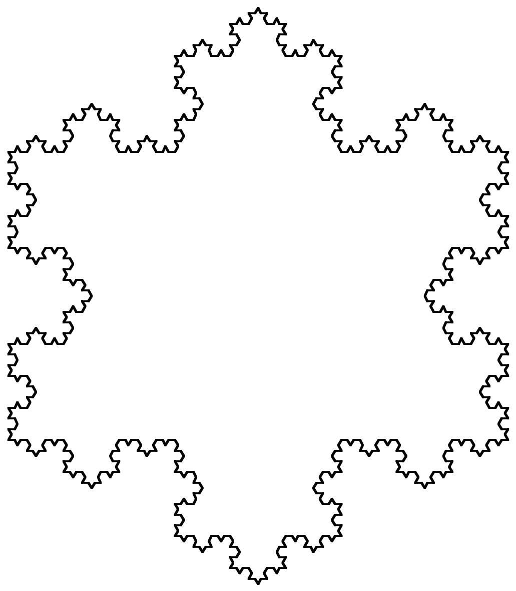
This particular grid was introduced in [2]. Earlier studies, such as [3], used a similar but finer grid which incorporated the vertices of the approximation of the boundary. It is then natural to enforce the boundary condition by simply requiring that $u(x) = 0$ at these vertices. The authors of [2] report higher accuracy using a larger spacing, thus the technique here might be thought of as more efficient.

2. Implementation

■ 2.1 Setting up the Laplacian approximation

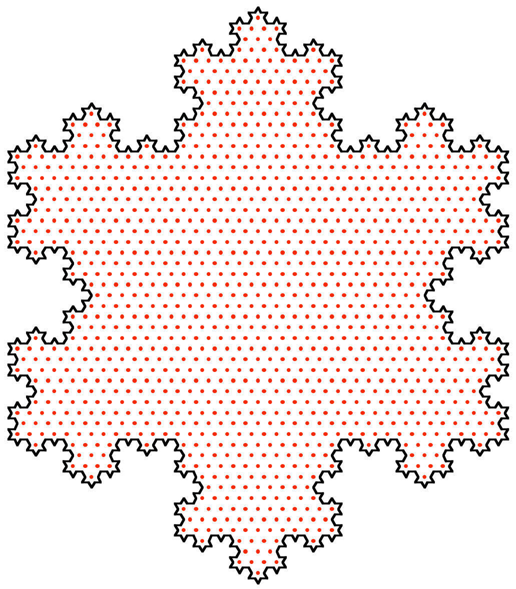
We begin by setting up the boundary of the snowflake. The level, **`boundaryLev`**, of this approximation is related to, but distinct from, the level of the approximation of the grid of interior points. Both are defined in terms of an overall variable, **`lev`**. The boundary is assembled using some fairly standard techniques for the generation of self-similar sets as described in, for example, [4].

```
lev = 3;
boundaryLev = lev + 1;
RotationMatrix[t_] := {{Cos[t], Sin[-t]}, {Sin[t], Cos[t]}};
KochStep[{p1_, p2_}] := With[{
    q1 = p1 + (p2 - p1) / 3,
    q2 = (p1 + (p2 - p1) / 3) + RotationMatrix[-Pi / 3].(p2 - p1) / 3,
    q3 = p1 + 2 (p2 - p1) / 3},
   {p1, q1, q2, q3, p2}];
KochStep[pp : {{_, _} ..}] := Join[Partition[Flatten[Most /@
      (KochStep /@ Partition[pp, 2, 1])], 2], {pp[[-1]]}];
KochVertices = Nest[KochStep, N@{{3 √3 / 4, 3 / 4}, {-3 √3 / 4, 3 / 4},
     {0, -3 / 2}, {3 √3 / 4, 3 / 4}}, boundaryLev];
ksfPic = Graphics[{Thickness[0.005], Line[KochVertices]},
   AspectRatio → Automatic];
Show[ksfPic];
```



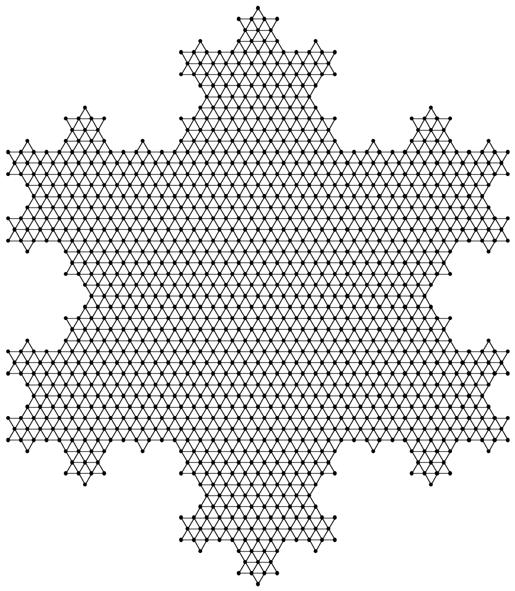
We use the self-similarity of the snowflake itself to generate the interior grid. This step is a bit trickier. The snowflake is self-similar consisting of seven copies of itself. More precisely, there are seven functions $f_0, f_1, \ldots, f_6$ that map the snowflake onto the constituent parts shown in figure 1. Take $x_0 = (0, 0)$ to be the zeroth level approximation to the interior grid. Any other point in the grid may be realized as $f_{i_1} \circ f_{i_2} \circ \cdots \circ f_{i_n}(x_0)$, for some sequence of the f_is. The interior grid is formed by all points generated by such a sequence so that the contraction ratio of $f_{i_1} \circ f_{i_2} \circ \cdots \circ f_{i_n}$ is just smaller than a pre-specified amount, namely $1 / 3^{\mathbf{drumLev}/2}$.

```
drumLev = 2 lev;
RotationMatrix[t_] := {{Cos[t], Sin[-t]},
   {Sin[t], Cos[t]}};
f0 = {RotationMatrix[Pi / 2] / √3 , {0, 0}};
A = {{1, 0}, {0, 1}} / 3;
ksfIFS = Prepend[Table[{A, {Cos[t], Sin[t]}},
    {t, Pi / 6, 11 Pi / 6, Pi / 3}], f0];
toFunc[{A_?MatrixQ, b_List}] := A.# + b &;
ksfFuncs = toFunc /@ ksfIFS;
f[v_List, i_] := ksfFuncs[[i + 1]][v]
r₀ = 1 / √3 ; rₙ_ = 1 / 3;
init = at[{}, 1];
seqs = init //. at[paths : {Integer___}, s_] :>
    Table[at[Prepend[paths, i], s * rᵢ], {i, 0, 6}] /;
     s > 1 / 3 ^ (drumLev / 2);
seqs = First /@ Flatten[seqs];
interiorApproxs = FoldList[f, {0., 0.}, #] & /@ seqs;
interiorGrid = Last /@ interiorApproxs;
pointPic = ListPlot[interiorGrid,
   DisplayFunction → Identity,
   PlotJoined → False];
flakeWithGridPic = Show[ksfPic, pointPic];
```



Given a point in this grid, we need quick access to its nearest neighbors in the grid. We can store this information in a nearest neighbor graph, represented as an adjacency matrix, and we can check that we have the correct graph using **`GraphPlot`**.

```
dist[h[p_], h[q_]] := Norm[p - q];
adj = SparseArray[Outer[dist[#1, #2] == (1.0 / 3) ^ ((drumLev - 1) / 2) &,
     h /@ interiorGrid, h /@ interiorGrid] /.
    {True → 1, False → 0}];
Needs["DiscreteMath`GraphPlot`"];
interiorGraph = GraphPlot[adj,
   VertexCoordinates → interiorGrid];
```



We can now set up the matrix which approximates the Laplacian using formula 0 and compute several eigenvalues. Use of the **Shift→0** option of the Arnoldi method finds smaller eigenvalues first, rather than larger eigenvalues. This way, we find the lower frequencies first. The **scale** is the linear scale of size between our Koch drum and the one in [2]; thus, we can compare results.

```
stepSize = (1.0 / 3) ^ ((drumLev - 1) / 2);
scale = ((3 / 2) / (√3 / 3));
laplacianRules = Join[{
    {i_, i_} :→ (Total[2 adj[[i]]] - 24) / (3 stepSize^2)},
   ArrayRules[adj] /. ({i_, j_} → 1) → ({i, j} → 2 / (3 stepSize^2))];
laplacianMatrix = SparseArray[laplacianRules];
Reverse[Eigenvalues[laplacianMatrix, 10,
   Method → {"Arnoldi", Shift → 0,
     Tolerance → 10^-16}]] * scale^2
```

```
{-39.3312, -97.1845, -97.1845, -164.603,
 -164.603, -189.195, -207.366, -269.958, -269.958, -308.947}
```

These numbers are close to those reported in [2], although the level could be increased to obtain more precise results. The repeated eigenvalues arise due to symmetries between the fundamental modes of vibration.

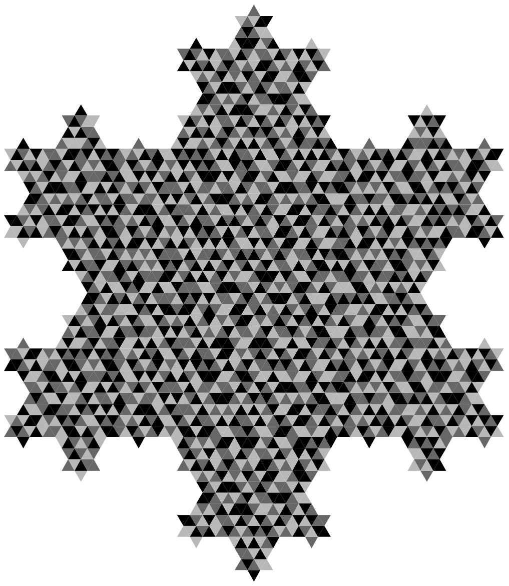
■ 2.2 Setting up for the graphics

Each vibrational mode of the drum may be generated from an eigenvector of our Laplacian matrix, since each component of the vector indicates the relative magnitude of the vibration at a particular point in the interior grid. To set up these graphics, we need to access the individual triangles in our decomposition of the snowflake. Given a vertex in the grid, it is easy to find the triangles containing that vertex, since these are determined by the vertex itself and two of its adjacent vertices. We don't want to do this for every vertex in the interior grid, however, since each triangle would be determined three times. The smaller grid **interiorGrid2** has the property that each triangle contains exactly one vertex from **interiorGrid2**. The points of **interiorGrid2** for a lower level decomposition are shaded gray in figure 0.

```
interiorGrid2 = Union[#[[-2]] & /@ interiorApproxs];
interiorGrid2Positions = Flatten[Position[interiorGrid, #] & /@ interiorGrid2];
adjLists = #[[1, 1]] & /@ Most[ArrayRules[#]] & /@ adj;
triangles[v_] := Module[{pairs},
   pairs = Flatten[Table[adjLists[[v]][[{i, j}]],
      {i, 1, Length[adjLists[[v]]]}, {j, i + 1, Length[adjLists[[v]]]}], 1];
   Join[{v}, #] & /@ Select[pairs, MemberQ[adjLists[[#[[1]]]], #[[2]]] &]];
triangleIndices = Flatten[triangles /@ interiorGrid2Positions, 1];
```

We can plot the triangles as a simple check to verify that we've achieved the desired partition.

```
Show[Graphics[MapIndexed[{GrayLevel[Mod[#2[[1]], 3] / 3],
      Polygon[interiorGrid[[#]]]} & , triangleIndices]],
  AspectRatio → Automatic];
```



A glance at figure 0 shows that the snowflake decomposes into a collection of (mostly) triangles together with quadrilaterals near the boundary. Accessing the quadrilaterals is a bit trickier and we will wait until we need to construct the final 3D image to do so.

■ 2.3 Generating 3D graphics

We still have a bit of work before we can generate 3D images of our Koch drum. One thing we will certainly need is the magnitude of the vibration at each interior grid point. These magnitudes are given by the eigenvectors of the Laplacian matrix. We compute several of those and store one of them to work with.

```
evs = Reverse[Eigenvectors[laplacianMatrix, 10,
    Method → {"Arnoldi", Shift → 0}]];
ev = evs[[6]];
```

We now turn our attention to constructing the portion of the drum over the quadrilaterals on the boundary of our decomposed snowflake. We first find the points from the interior grid that have fewer than six neighbors; these are the points nearest to the boundary. The vertices of the quadrilaterals are chosen from these points together with points from the boundary itself, stored in **KochVertices**. A further complication is the occurrence of two types of quadrilaterals - acute and obtuse. Note that each acute quadrilateral occurs at an acute interior angle of the boundary. Furthermore, the sequence of angles in the boundary occurs in a predictable pattern, which we store in **anglePattern**.

```
nearBoundaryPositions = Flatten[
   Position[Total /@ adj, _?(# < 6 &)]];
anglePattern = Flatten[Nest[# /. {ac → {ac, ob, ac, ob},
       ob → {ob, ob, ac, ob}} &, {ac, ac, ac}, boundaryLev]];
```

We can use this to efficiently assemble the quadrilaterals. The individual quadrilaterals will be computed using the functions **acuteQuad3D** and **obtuseQuad3D**.

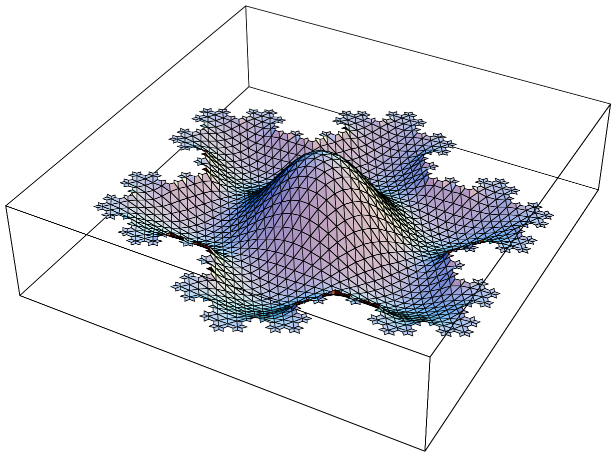
```
acuteQuad3D[n_, ev_] := Module[
   {interiorIndex, oneFudge},
   If[n == 1, oneFudge = -2, oneFudge = n - 1];
   interiorIndex = Select[nearBoundaryPositions,
      Norm[interiorGrid[[#]] - KochVertices[[n]]] < stepSize &][[1]];
   Polygon[{Join[KochVertices[[n]], {0}], Join[KochVertices[[n + 1]], {0}],
     Join[interiorGrid[[interiorIndex]], {ev[[interiorIndex]]}],
     Join[KochVertices[[oneFudge]], {0}]}]];
obtuseQuad3D[n_, ev_] := Module[
   {interiorIndex1, interiorIndex2},
   interiorIndex1 = Select[nearBoundaryPositions,
      Norm[interiorGrid[[#]] - KochVertices[[n]]] ≤ stepSize / 2 &][[1]];
   interiorIndex2 = Select[nearBoundaryPositions,
      Norm[interiorGrid[[#]] - KochVertices[[n + 1]]] ≤ stepSize / 2 &][[1]];
   Polygon[{Join[KochVertices[[n]], {0}], Join[KochVertices[[n + 1]], {0}],
     Join[interiorGrid[[interiorIndex2]], {ev[[interiorIndex2]]}],
     Join[interiorGrid[[interiorIndex1]], {ev[[interiorIndex1]]}]}]];
```

We use **MapAt** in conjunction with the list of angle patterns to construct the right type of quadrilateral at each location.

```
acuteQuads3D = Cases[MapAt[acuteQuad3D[#, ev] &, Range[Length[anglePattern]],
    Position[anglePattern, ac]], _Polygon];
obtuseQuads3D = Cases[MapAt[obtuseQuad3D[#, ev] &, Range[Length[anglePattern]],
    Position[Partition[anglePattern, 2, 1], {ob, ob}]], _Polygon];
```

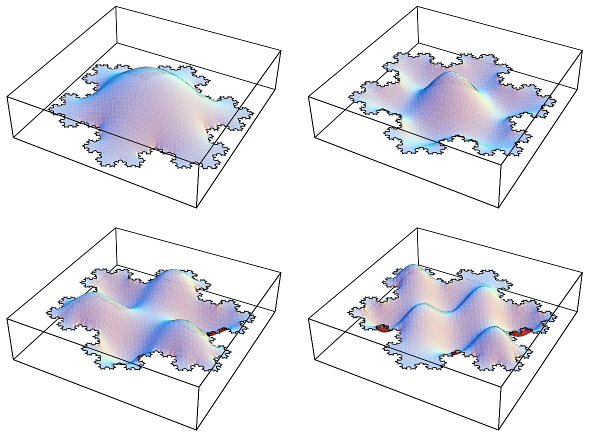
We now proceed in very much the same way that we did when we built our test graphic, but we want to generate 3D polygons incorporating the eigenvector, rather than 2D polygons.

```
triangle3D[{v1_, v2_, v3_}] := Polygon[Flatten /@ {v1, v2, v3}]
allTriangles3D = triangle3D[Transpose[{interiorGrid[[#]], ev[[#]]}]] & /@
   triangleIndices;
Show[Graphics3D[{allTriangles3D, obtuseQuads3D, acuteQuads3D}],
  PlotRange → All, BoxRatios → {1, 1, 1 / 4}];
```



We can write a function **evPlot** that incorporates all of these ideas to generate an array of these pictures. Note that **evPlot** is not stand-alone; it uses much information that we have already computed. Also, **evPlot** uses **EdgeForm[]** to suppress the mesh.

```
evPlot3D[ev_, opts___] := Module[
   {acuteQuads3D, obtuseQuads3D,
    allTriangles3D, perim},
   acuteQuads3D = Cases[MapAt[acuteQuad3D[#, ev] &, Range[Length[anglePattern]],
      Position[anglePattern, ac]], _Polygon];
   obtuseQuads3D = Cases[MapAt[obtuseQuad3D[#, ev] &, Range[Length[anglePattern]],
      Position[Partition[anglePattern, 2, 1], {ob, ob}]], _Polygon];
   allTriangles3D =
    triangle3D[Transpose[{interiorGrid[[#]], ev[[#]]}]] & /@ triangleIndices;
   perim = Line[Join[#, {0}] & /@ KochVertices];
   Show[Graphics3D[{EdgeForm[], allTriangles3D,
      obtuseQuads3D, acuteQuads3D, perim}],
    opts]];
pics = evPlot3D[#, DisplayFunction → Identity,
     PlotRange → All, BoxRatios → {1, 1, 1 / 4}] & /@ evs[[{1, 6, 7, 10}]];
Show[GraphicsArray[Partition[pics, 2]]];
```



References

[1] M. Berry, in W. Guttinger and H. Elkheimer, (Eds.) *Structural Stability in Physics*, Springer-Verlag, Berlin. “Distribution of Modes in Fractal Resonators” (1979) 51–53.

[2] J. Neuberger, N. Sieben, and J. Swift, “Computing eigenfunctions on the Koch Snowflake: a new grid and symmetry.” *J. Comp. Appl. Math.* **191** (2006) 126-142.

[3] M. Lapidus, “Fractal drum, inverse spectral problems for elliptic operators and a partial resolution of the Weyl-Berry conjecture.” *Trans. Amer. Math. Soc.* **325** 465-529.

[4] M. McClure, “Directed-graph iterated function systems.” *Math. Educ. Res.* **9** (2000).