Cantor shadows (using Iterated Function Systems and Regions)

GONZALO GARCÍA ALARCÓN ESTRADA, **********
Dec / 22 / 2017
University of Toronto, Shameless Mathematica

Shwadows of the Cantor square:

Draw an approximation of the Cantor square C2, rotate it by an angle θ and project the rotation vertically, draw the projection, and compute its measure. Manipulate.

PART 1: using Iterated Function Systems (IFS) whith built in transformations and projections... (a bit slow)

[NO changes in PART I since last time]

DECLARATION OF FUNCTIONS

```
(* initial square *)
(* R=ImplicitRegion[0≤ x≤ 1∧0≤ y≤ 1,{x,y}]; *)
r = Rectangle[];
```

```
(* transformations *)
t1 = AffineTransform[IdentityMatrix[2] * (1/3)];
t2 = AffineTransform[{IdentityMatrix[2] * (1/3), {0, 2/3}}];
t3 = AffineTransform[{IdentityMatrix[2] * (1/3), {2/3, 2/3}}];
t4 = AffineTransform[{IdentityMatrix[2] * (1/3), {2/3, 0}}];
T[r_] := RegionUnion[TransformedRegion[r, t1],
   TransformedRegion[r, t2], TransformedRegion[r, t3], TransformedRegion[r, t4]];
rota[r_, θ_] := TransformedRegion[r, RotationTransform[θ]];
(* rotarad[r_,θ_]:=TransformedRegion[r,RotationTransform[θ ]]; *)
cantorN[r_, n_Integer?NonNegative] := Nest[T, r, n];
```

```
(* projections *)
Proj[r_] := Resolve[∃y {x, y} ∈ r, Reals] // Reduce; (* project *)
r2interval[r_] := Module[{p = Proj[r]}, Switch[Head[p],
    Or,
    Module[{aux}, aux = List @@@ List @@ p /. {a_, LessEqual, x, LessEqual, b_} :> {a, b};
     Return[Interval @@ aux]],
    Inequality,
    Module[{aux},
     aux = List @@ p /. {a_, LessEqual, x, LessEqual, b_} :> {a, b};
     Return[Interval@aux]]
   ]];
(* transform into an interval that Mathematica understands *)
```

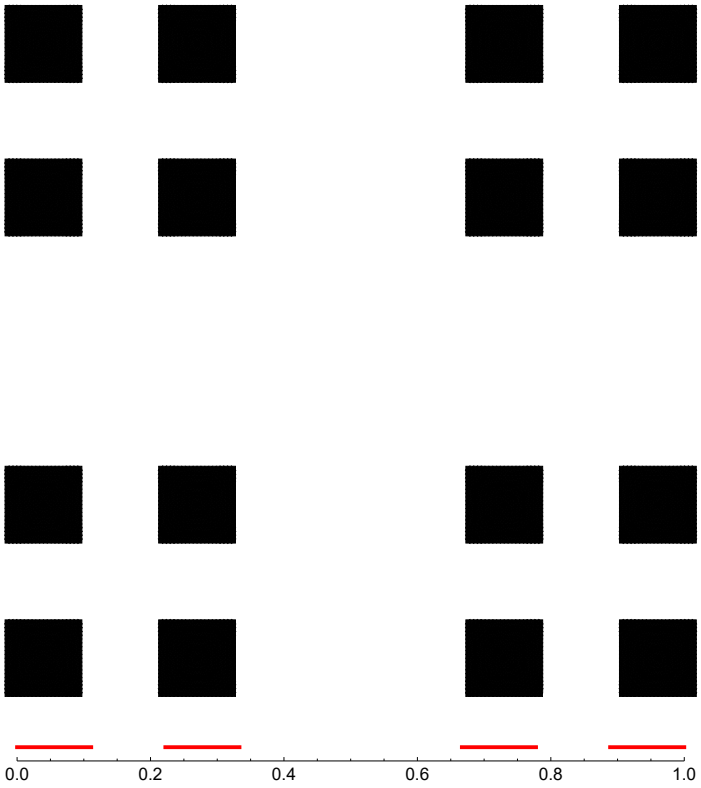
```
(* plotting *)
Disp[r_] := RegionPlot[r, PlotRange → RegionBounds[r],
   BoundaryStyle → None, PlotStyle → Black, Frame → False];
Disp[r_Interval] := NumberLinePlot[r, {x, Min[r], Max[r]},
   PlotStyle → {PointSize[0], Red, Thick}, PlotRange → {{Min[r], Max[r]}, Automatic}];
pos[r_] := TransformedRegion[r, TranslationTransform[{0, 0.5}]];

Shadow[n_Integer?NonNegative, ∂_](* Takes in DEGREES!!!*) := Module[{set, proj},
  If[(n + 1) > Length[C2], Return["n is too big"],
   set = rota[C2[[n + 1]], ∂Degree];
   proj = set // r2interval;
   Return[
    Show[set // pos // Disp, proj // Disp, lab[set] // Graphics, lab[proj] // Graphics]]]
 ]
(* text in figures *)
lab[r_] := Text[
   Style["Measure of set: " <> formatlab[r], Gray, FontSize → 13], Scaled[{0.17, .6}]];
lab[r_Interval] := Text[Style["Measure of projection: " <> formatlab[r],
    Red, FontSize → 13], Scaled[{0.22, 0.55}]];
formatlab[r_] := If[Head[RegionMeasure[r]] === Rational,
  RegionMeasure[r] /. Rational[p_, q_] :> ToString[p] <> "/" <> ToString[q],
  ToString[RegionMeasure[r] // N]
 ]
```

```
(* pre-generate cantor squares, as REGIONS *)
i = 4; (*number of iterations*)
C2 = NestList[T, r, i];
```

FIRST : AN EXAMPLE (directly generated, not pre-generated)

```
C2N = cantorN[r, 2]; C2Nint = C2N // r2interval;
```

```
a = C2N // pos // Disp
b = C2Nint // Disp
C2Nint
C2Nint // RegionMeasure
Show[lab[C2N] // Graphics, lab[C2Nint] // Graphics]
```



$$\text{Interval}\left[\left\{0, \frac{1}{9}\right\}, \left\{\frac{2}{9}, \frac{1}{3}\right\}, \left\{\frac{2}{3}, \frac{7}{9}\right\}, \left\{\frac{8}{9}, 1\right\}\right]$$

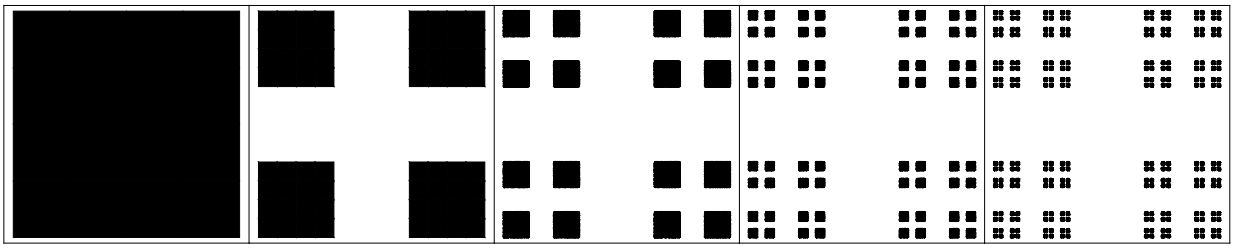
$$\frac{4}{9}$$

Measure of set: 16/81

Measure of projection: 4/9

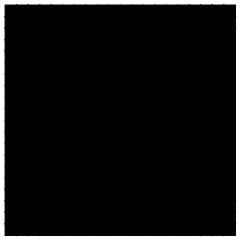
AN APPROXIMATION TO THE CANTOR SQUARE

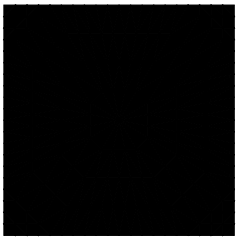
```
GraphicsRow[Graphics /@ Disp /@ C2, Frame → All, ImageSize → Full]
```



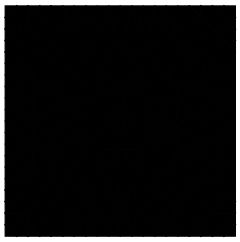
ROTATIONS and PROJECTIONS

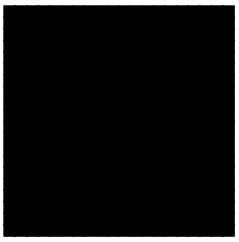
```
Shadow[1, 0] (* n,θ ... takes θ in Degrees *)
```



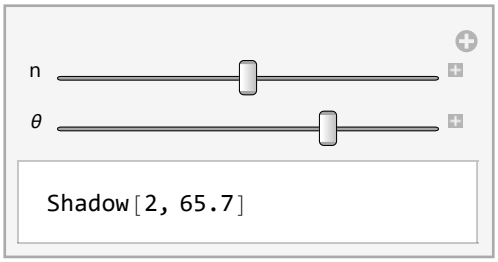


Measure of set: 4/9
Measure of projection: 2/3





```
Manipulate[Quiet[Shadow[n, θ]], {{n, 0}, 0, Length[C2] - 1, 1}, {{θ , 0}, 0 , 90 }]
```



PART 2: using binary expressions and my own interval arithmetic and projections... MUCH FASTER!

(based on the code by Prof. D. Bar-Natan: http://drorbn.net/AcademicPensieve/Classes/17-1750-ShamelessMathematica/nb/171201-CantorGames.pdf)

NOTE: The green cells are essentially the code, i.e. the functions used, the rest are examples to illustrate each step. I decided to leave them this time since I think it illustrates the process much more clearly, as opposed to just showing the final result.
The orange cells contain code to make the plots look nice.

```
(* rotation matrix *)
Rot[θ_] := {{Cos[θ], -Sin[θ]}, {Sin[θ], Cos[θ]}};
(* left end of intervals,
from the binary expression of an interval on level n of the Cantor set*)
left[binary_List] := binary.Table[2/3^k, {k, Length[binary]}];
```

```
CantCorners[n_Integer?NonNegative] := Flatten@Table[
   C@{left[binx], left[biny]},
   {binx, Tuples[{0, 1}, n]}, {biny, Tuples[{0, 1}, n]}
  ]
```

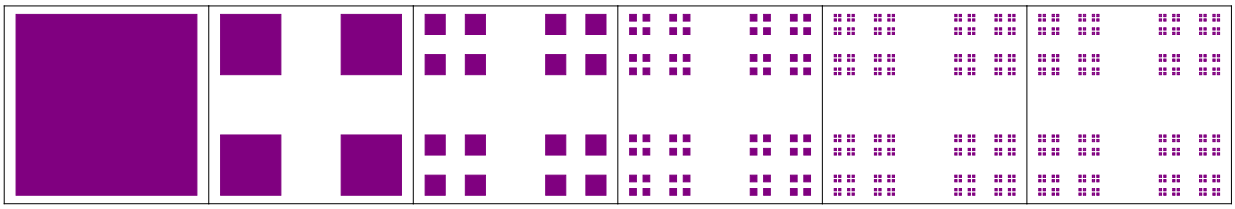
```
CantCorners[1]
```

$$\left\{C[\{0,0\}],\ C\left[\left\{0,\frac{2}{3}\right\}\right],\ C\left[\left\{\frac{2}{3},0\right\}\right],\ C\left[\left\{\frac{2}{3},\frac{2}{3}\right\}\right]\right\}$$

```
(* rotated Cantor Square *)(* translated to hace (0,0) at center *)
CantPoly[n_Integer?NonNegative, θ_] :=
 CantCorners[n] /. C[p_List] :> Polygon[{p - {0.5, 0.5}, p + {1/3^n, 0} - {0.5, 0.5},
      p + {1/3^n, 1/3^n} - {0.5, 0.5}, p + {0, 1/3^n} - {0.5, 0.5}}.Rot[θ]]
```

AN APPROXIMATION TO THE CANTOR SQUARE (V2)

```
GraphicsRow[Map[Graphics[{Purple, CantPoly[#, 0]}] &, Range[6] - 1],
 Frame → All, ImageSize → Full]
```



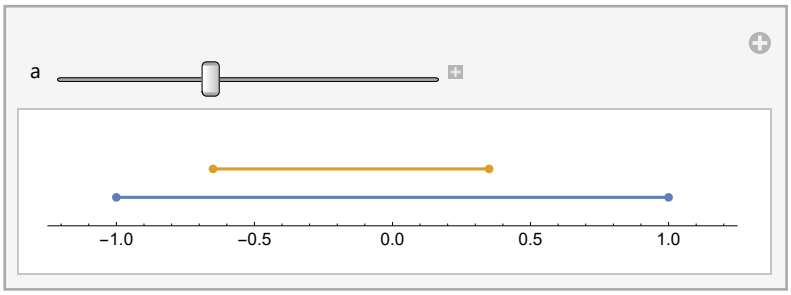
ROTATIONS and PROJECTIONS (V2)

```
CantPoly[1, 0]
```

```
{Polygon[{{-0.5, -0.5}, {-0.166667, -0.5}, {-0.166667, -0.166667}, {-0.5, -0.166667}}],
 Polygon[{{-0.5, 0.166667}, {-0.166667, 0.166667}, {-0.166667, 0.5}, {-0.5, 0.5}}],
 Polygon[{{0.166667, -0.5}, {0.5, -0.5}, {0.5, -0.166667}, {0.166667, -0.166667}}],
 Polygon[{{0.166667, 0.166667}, {0.5, 0.166667}, {0.5, 0.5}, {0.166667, 0.5}}]}
```

```
(* boolean DISJOINT INTERVALS TEST: True = disjoint *)
Disj[Int[a1_, a2_], Int[b1_, b2_]] :=
 If[((b1 ≤ a2) ∧ (a1 ≤ b2)) ∨ ((b1 ≤ a1) ∧ (a2 ≤ b2)), False, True]
```

```
(* dynamic example of disjoint test *)
Manipulate[
 color = If[Disj[Int[-1, 1], Int[a, a + 1]], Purple, Cyan];
 NumberLinePlot[
  {Int[-1, 1], Int[a, a + 1]} /. Int[a_, b_] :> Interval[{a, b}], PlotStyle → color]
 ,
 {a,
  -3,
  3}]
```




```
(* projection *)
Proj[cant_List] := Module[{p1, p2, p3, p4},
  p1 = cant /. Polygon[{{a_, _}, {b_, _}, {c_, _}, {d_, _}}] :>
     Int[Min[a, b, c, d], Max[a, b, c, d]];
  p2 = Sort[Times @@ p1] /. Int[a_, b_]^n_ :> Int[a, b];
  (* interval union *)
  p3 = p2 //.
    Int[a_, b_] Int[c_, d_] :> Int[Min[a, b], Max[b, d]] /; ! Disj[Int[a, b], Int[c, d]];
  (* eliminate square bug *)
  p4 = p3 /. Int[a_, b_]^n_ :> Int[a, b]
 ]
```

```
CantPoly[2, 5. Degree] // Proj
```

```
Int[-0.541675, -0.343831] Int[-0.320299, -0.122455]
 Int[0.122455, 0.320299] Int[0.343831, 0.541675]
```

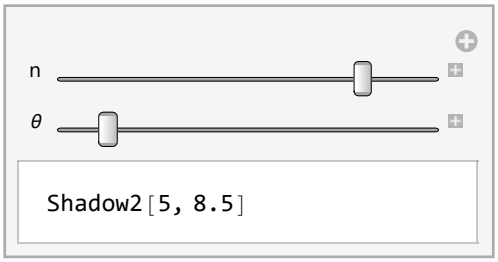
```
(* measure of projection *)
ProjMeas[p_Times] := Plus @@ p /. Int[a_, b_] :> b - a;
ProjMeas[Int[a_, b_]] := b - a;
(* measure of Cantor square *)
CantMeas[n_Integer?NonNegative] := (4^n) (1/3^n)^2;
```

```
CantPoly[2, 5. Degree] // Proj // ProjMeas
```

```
0.791376
```

```
Disp[p_Times] :=
  Graphics[{Red, Thick, (List @@ p) /. Int[a_, b_] :> Line[{{a, 0}, {b, 0}}]}];
Disp[p_Int] := Graphics[{Red, Thick, p /. Int[a_, b_] :> Line[{{a, 0}, {b, 0}}]}];

lab2[n_Integer] := Text[Style["Measure of set: " <>
      (CantMeas[n] /. Rational[p_, q_] :> ToString[p] <> "/" <> ToString[q]),
    Gray, FontSize → 13], Scaled[{0.17, .6}]];
lab2[0] := Text[Style["Measure of set: 1", Gray, FontSize → 13], Scaled[{0.17, .6}]];
lab2[p_Times] :=
  Text[Style["Measure of projection: " <> ToString[ProjMeas[p]], Red, FontSize → 13],
   Scaled[{0.22, 0.55}]];
lab2[p_Int] := Text[Style["Measure of projection: " <> ToString[ProjMeas[p]],
    Red, FontSize → 13], Scaled[{0.22, 0.55}]];

Shadow2[n_Integer?NonNegative, θ_]
 (* Takes in DEGREES!!!*) := Module[{set, proj, color},
  color = ColorData["Rainbow"];
  set = CantPoly[n, θDegree];
  proj = set // Proj;
  Return[Show[
    Graphics[{color[θ/90], set}],
    proj // Disp,
    lab2[n] // Graphics,
    lab2[proj] // Graphics,
    ImageSize → 420
   ]]
 ]
```

```
Manipulate[Shadow2[n, θ], {{n, 0}, 0, 6, 1}, {{θ , 0}, 0 , 90 }]
```



The key to understand the behaviour of the projection, and why we recover a single interval, is to analyze the behaviour of the first iterative step, i.e. n=1.

For n=1, for a $\theta \in (18^o,19^o) \cup (71^o,72^o)$ there is a transition from having 2 intervals to 1 inteval in the projection. Note that for n=2, we get the same behaviour in each of the corners, but a smaller scale, hece for a smaller angle. For n=2, once we transition from 4 to 2 intervals due to the same bahiour as for n=1, we are in a situation in which each of the 4 squares in each corner behave as one of the bigger squares for n=1. We therefore return to the situation of n=1.

This way, as n increases, we get the same situation as for n=1 but in smaller and smaller scales which "cascade" to reduce to the n=1 situation. Just as for n=2, the transition from n to n-1 happens at an angle smaller and smaller as n increases. This behaviour can easily be seen by playing with the Manipulate above.

These are only finite aproximations to the Cantor Square. In the limit as $n \to \infty$, we no longer have squares that project to intervals when there is no rotation, but points of measure zero which projet to points (actually a cantor set) when there is no projection. My intuition says that the same behaviour would be recovered, as when n grows, the increase in θ to transfer from n to n-1 is smaller, hence at the limit is sounds that with any small amount of rotation the points would align to project to an interval and recover the same behaviour as with the approximations. Therefore I claim that for the Cantor Square rotated for $\theta \in [19^o,71^o]$, the projection is a single interval.