# “Art of Science” competition

Justin Pearson
2015-02-19

Competition organized through the Center for Science and Engineering Partnerships at the University of California at Santa Barbara.

## Summary

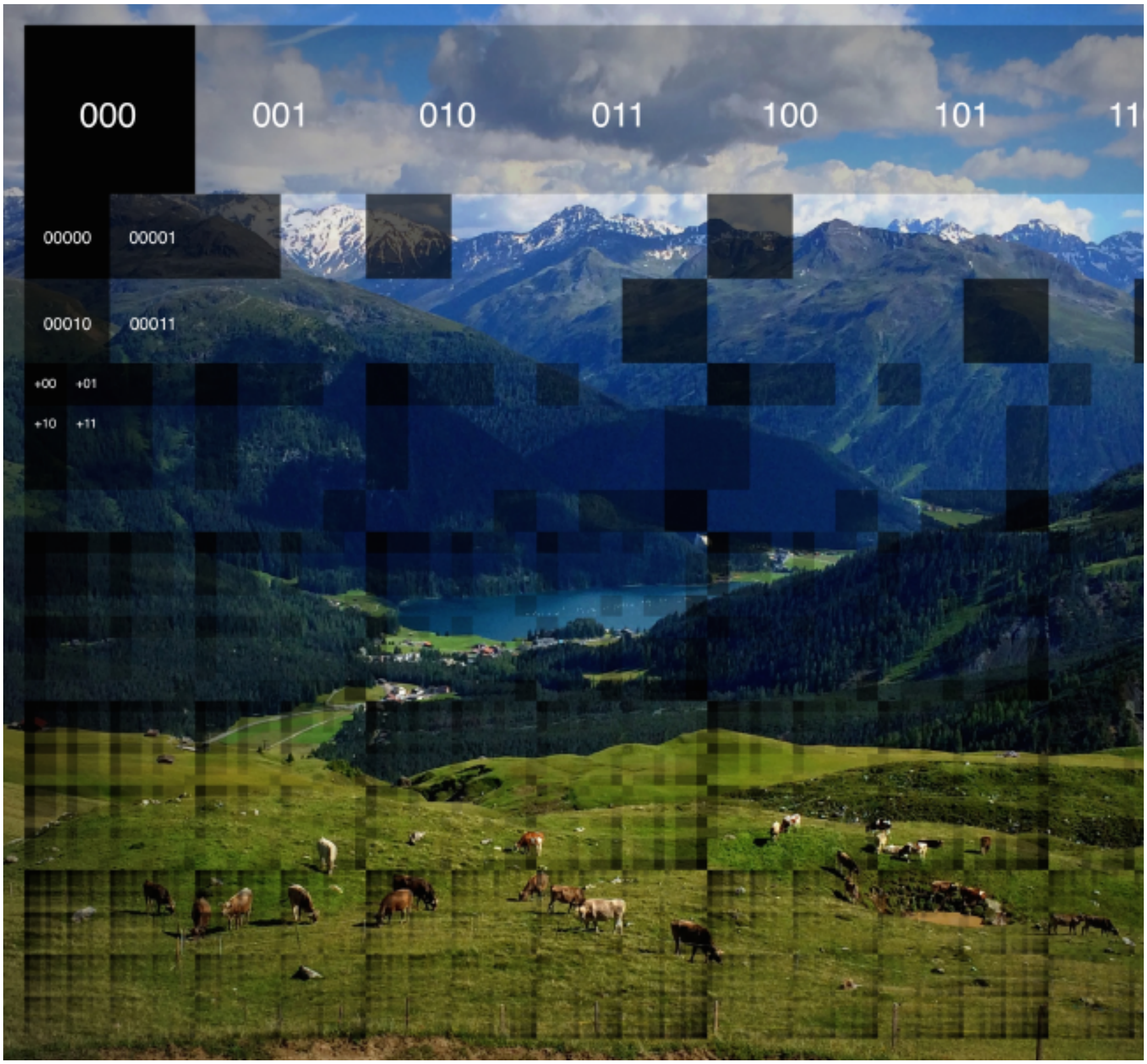
We stream movies, send pictures to friends, and video-chat with distant loved ones, all digitally, and all without a second thought. Empowering this revolution behind the scenes is Information Theory, which provides a mathematical framework to quantify, compress, and transmit information.

This picture illustrates an important theorem in Information Theory: the Asymptotic Equipartition Property. It formalizes and generalizes the intuitive notion that if you flip a fair coin many times, you would expect about 50% heads. In the image, each square represents a string of coinflips (with 0=tails and 1=heads), with smaller squares representing longer strings of flips. Like a family tree, each square recursively generates 4 squares below it by appending one of 4 suffixes: 00, 01, 10, or 11. Each square is black, but is made transparent depending on how close to “50% heads” its corresponding string of coinflips is. We see that the vast majority of the tiny squares at the bottom are nearly 50% heads and hence transparent, allowing the underlying Swiss pasture scene to show through.

## Final picture

In[1]:=
```
Clear["Global`*"];
SetDirectory[NotebookDirectory[]];
Thumbnail["art-csep-pearson.jpg", 800]
```

Out[3]=



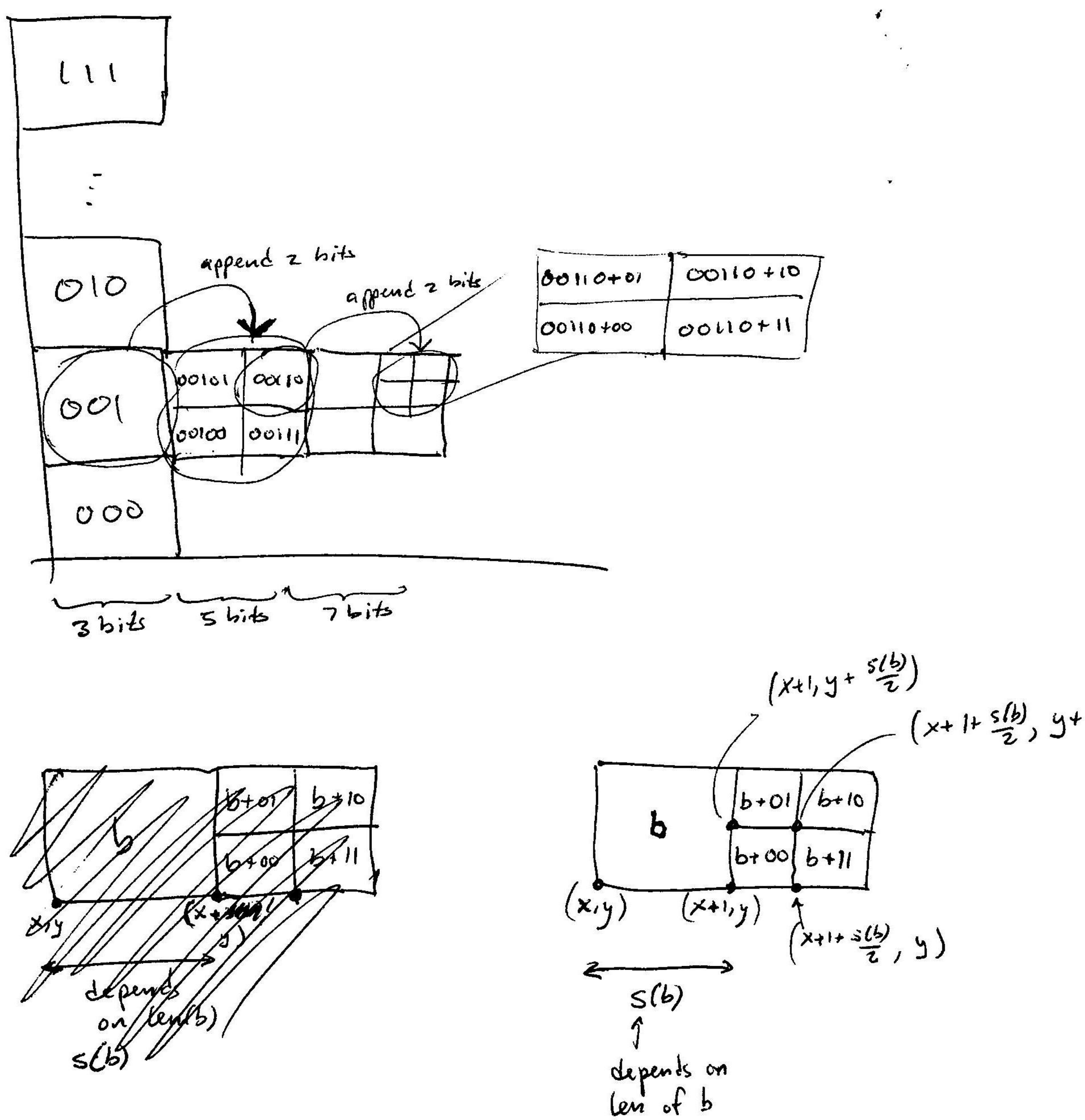

# How it's made

## Basic idea

Each square represents a bitstring. Every 'child' square inherits the parent's bitstring with +00, +01, +10, or +11 added.

In[4]:= `Show[Import["idea.jpg"], ImageSize → 700]`

Out[4]=




## Define object: ‘square’

We represent a bitstring as a ‘square’ object *square[ bitstring, position, color ]*.

This defines a bunch of functions on a ‘square’ object:

In[5]:=
```
Clear[square, bits, pos, color, sidelen, graphics]
square /: bits[sq_square] := sq[[1]]
square /: pos[sq_square] := sq[[2]]
square /: color[sq_square] := sq[[3]]
square /: sidelen[sq_square] := 2^-((Length[bits[sq]]-3)/2)
square /: graphics[sq_square] := Graphics[{
   {EdgeForm[Black], color[sq], Rectangle[pos[sq], pos[sq] + sidelen[sq] * {1, 1}]},
   Text[Style[bits[sq], FontSize → 20 * sidelen[sq]],
    pos[sq] + sidelen[sq]/2 * {1, 1}]}]
SetAttributes[graphics, Listable]
```

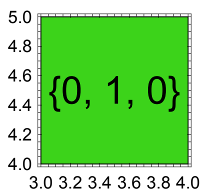
Example square:

In[12]:=
```
Clear[s]
s = square[{0, 1, 0}, {3, 4}, RandomColor[]]
```

Out[13]=
```
square[{0, 1, 0}, {3, 4}, ■]
```

In[14]:=
```
Show[graphics[s], Frame → True, ImageSize → 100]
```

Out[14]=




## Child squares.

Each bitstring can generate 4 ‘child’ bitstrings by append 00, 01, 10, or 11. Each of these bitstrings’ squares has a position based on the parent’s.

Here is a square for the bitstring “000” at position {0,0} with color Red:

In[15]:=
```
sq = square[{0, 0, 0}, {0, 0}, Red]
```

Out[15]=
```
square[{0, 0, 0}, {0, 0}, ■]
```

The possible suffixes:

In[16]:=
```
suffs = {{0, 0}, {1, 0}, {0, 1}, {1, 1}}
```

Out[16]=
```
{{0, 0}, {1, 0}, {0, 1}, {1, 1}}
```

Append each suffix to form the child squares:

In[17]:=
```
childbits = Table[bits[sq] ~Join~ suf, {suf, suffs}]
```

Out[17]=
```
{{0, 0, 0, 0, 0}, {0, 0, 0, 1, 0}, {0, 0, 0, 0, 1}, {0, 0, 0, 1, 1}}
```

Child squares’ positions are based on their parent position. They’re shifted over horizontally and then shifted around based on their suffix.

In[18]:=
```
childpos = Table[pos[sq] + {1, 0} + xy, {xy, sidelen[sq]/2 * suffs}]
```
Out[18]= $\left\{\{1, 0\}, \left\{\frac{3}{2}, 0\right\}, \left\{1, \frac{1}{2}\right\}, \left\{\frac{3}{2}, \frac{1}{2}\right\}\right\}$

For now, child colors are just lighter versions of the parent square:

In[19]:=
```
childcolors = Table[Lighter[color[sq]], Length[suffs]]
```
Out[19]= {■, ■, ■, ■}

Create the children:

In[20]:=
```
children = MapThread[square, {childbits, childpos, childcolors}];
Column[children]
```
Out[21]=

square[{0, 0, 0, 0, 0}, {1, 0}, ■]

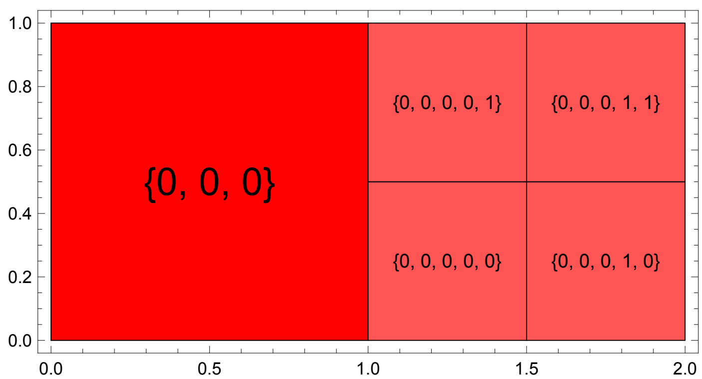
square[{0, 0, 0, 1, 0}, $\left\{\frac{3}{2}, 0\right\}$, ■]

square[{0, 0, 0, 0, 1}, $\left\{1, \frac{1}{2}\right\}$, ■]

square[{0, 0, 0, 1, 1}, $\left\{\frac{3}{2}, \frac{1}{2}\right\}$, ■]

The ‘graphics’ function we defined for squares is listable, so can be called on a list of children. Here is what the children look like:

In[22]:=
```
Show[{graphics[{sq, children}]}, Frame → True]
```
Out[22]=



## Wrap into a function.

In[23]:=
```
makeChildren[sq_square] :=
 Module[{s, suffs, childbits, childpos, children, childcolors},
 suffs = {{0, 0}, {1, 0}, {0, 1}, {1, 1}};
  s = sidelen[sq];
  childbits = Table[bits[sq] ~ Join ~ suf, {suf, suffs}];
  childpos = Table[pos[sq] + {1, 0} + xy, {xy, s/2 * suffs}];
  childcolors = Table[Lighter@color[sq], Length[childbits]];
  children = MapThread[square, {childbits, childpos, childcolors}];
  Return[children];
 ]
SetAttributes[makeChildren, Listable]
```

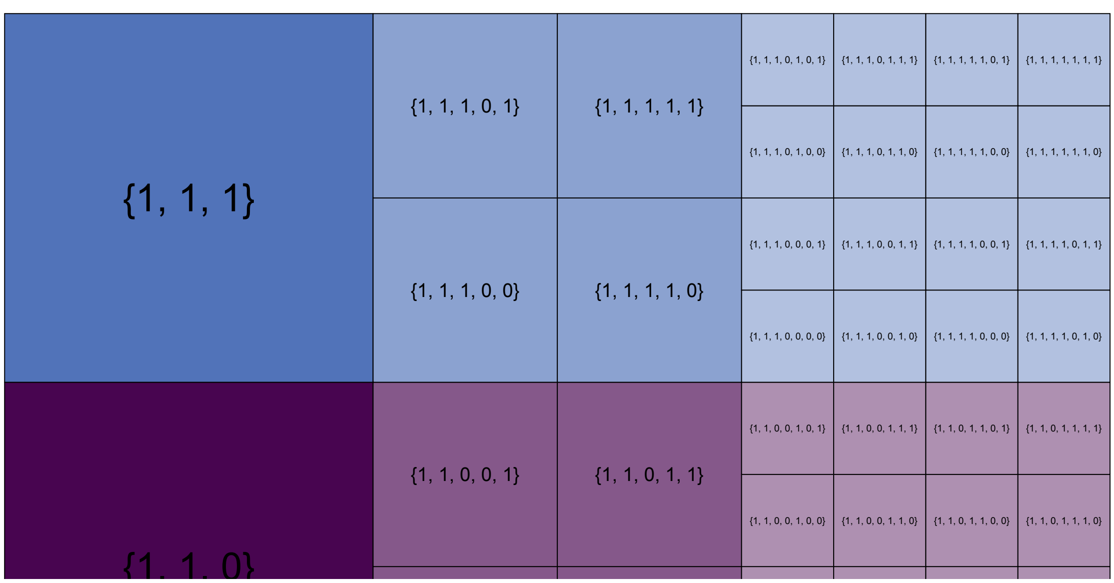
## Example use:

In[25]:=
```
initials = Table[square[IntegerDigits[i, 2, 3], {0, i}, RandomColor[]], {i, 0, 7}];
Column[initials]
```

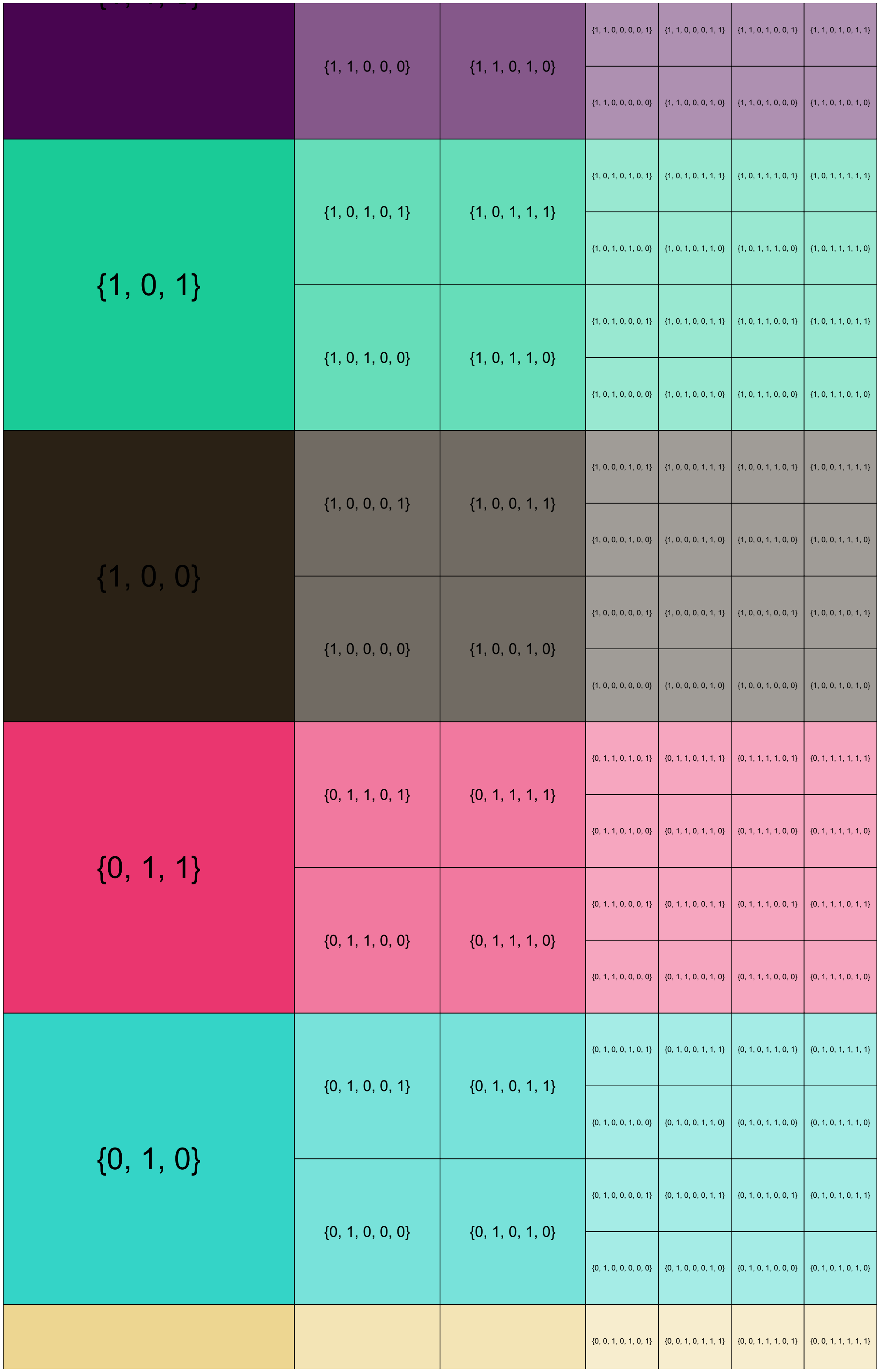
Out[26]=
```
square[{0, 0, 0}, {0, 0}, ]
square[{0, 0, 1}, {0, 1}, ]
square[{0, 1, 0}, {0, 2}, ]
square[{0, 1, 1}, {0, 3}, ]
square[{1, 0, 0}, {0, 4}, ]
square[{1, 0, 1}, {0, 5}, ]
square[{1, 1, 0}, {0, 6}, ]
square[{1, 1, 1}, {0, 7}, ]
```

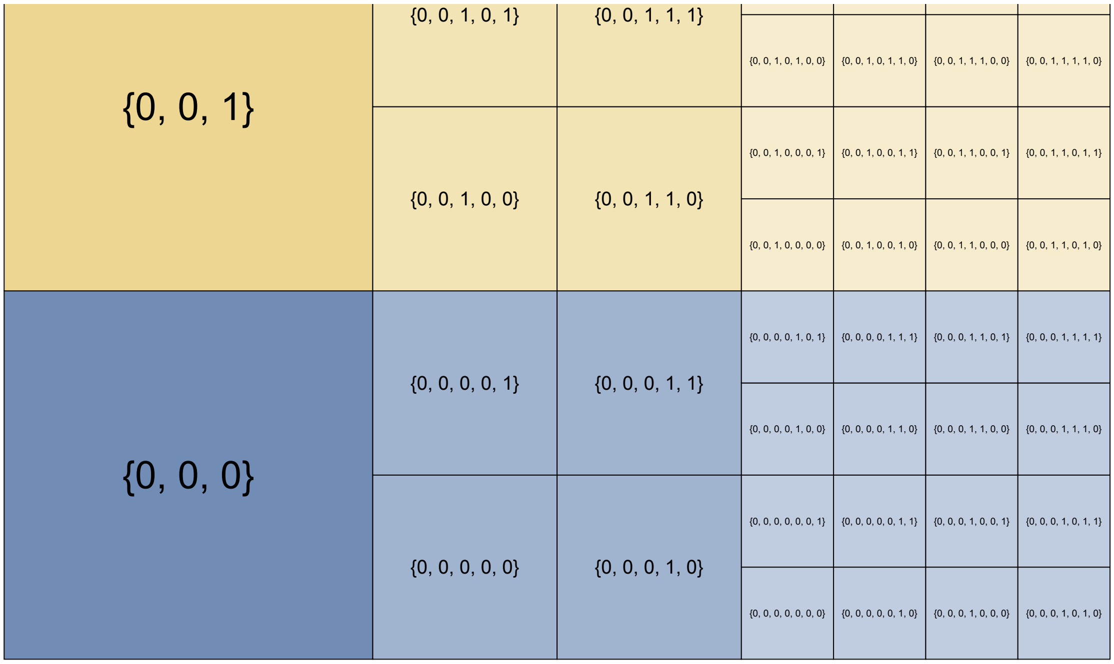
In[27]:=
```
allSquares = NestList[makeChildren, initials, 2];
```

In[28]:=
```
Show[graphics[allSquares], ImageSize → 600]
```



Out[28]=


{1, 1, 0, 0, 0}
{1, 1, 0, 1, 0}
{1, 1, 0, 0, 0, 0, 1}
{1, 1, 0, 0, 0, 1, 1}
{1, 1, 0, 1, 0, 0, 1}
{1, 1, 0, 1, 0, 1, 1}
{1, 1, 0, 0, 0, 0, 0}
{1, 1, 0, 0, 0, 1, 0}
{1, 1, 0, 1, 0, 0, 0}
{1, 1, 0, 1, 0, 1, 0}
{1, 0, 1}
{1, 0, 1, 0, 1}
{1, 0, 1, 1, 1}
{1, 0, 1, 0, 0}
{1, 0, 1, 1, 0}
{1, 0, 1, 0, 1, 0, 1}
{1, 0, 1, 0, 1, 1, 1}
{1, 0, 1, 1, 1, 0, 1}
{1, 0, 1, 1, 1, 1, 1}
{1, 0, 1, 0, 1, 0, 0}
{1, 0, 1, 0, 1, 1, 0}
{1, 0, 1, 1, 1, 0, 0}
{1, 0, 1, 1, 1, 1, 0}
{1, 0, 1, 0, 0, 0, 1}
{1, 0, 1, 0, 0, 1, 1}
{1, 0, 1, 1, 0, 0, 1}
{1, 0, 1, 1, 0, 1, 1}
{1, 0, 1, 0, 0, 0, 0}
{1, 0, 1, 0, 0, 1, 0}
{1, 0, 1, 1, 0, 0, 0}
{1, 0, 1, 1, 0, 1, 0}
{1, 0, 0}
{1, 0, 0, 0, 1}
{1, 0, 0, 1, 1}
{1, 0, 0, 0, 0}
{1, 0, 0, 1, 0}
{1, 0, 0, 0, 1, 0, 1}
{1, 0, 0, 0, 1, 1, 1}
{1, 0, 0, 1, 1, 0, 1}
{1, 0, 0, 1, 1, 1, 1}
{1, 0, 0, 0, 1, 0, 0}
{1, 0, 0, 0, 1, 1, 0}
{1, 0, 0, 1, 1, 0, 0}
{1, 0, 0, 1, 1, 1, 0}
{1, 0, 0, 0, 0, 0, 1}
{1, 0, 0, 0, 0, 1, 1}
{1, 0, 0, 1, 0, 0, 1}
{1, 0, 0, 1, 0, 1, 1}
{1, 0, 0, 0, 0, 0, 0}
{1, 0, 0, 0, 0, 1, 0}
{1, 0, 0, 1, 0, 0, 0}
{1, 0, 0, 1, 0, 1, 0}
{0, 1, 1}
{0, 1, 1, 0, 1}
{0, 1, 1, 1, 1}
{0, 1, 1, 0, 0}
{0, 1, 1, 1, 0}
{0, 1, 1, 0, 1, 0, 1}
{0, 1, 1, 0, 1, 1, 1}
{0, 1, 1, 1, 1, 0, 1}
{0, 1, 1, 1, 1, 1, 1}
{0, 1, 1, 0, 1, 0, 0}
{0, 1, 1, 0, 1, 1, 0}
{0, 1, 1, 1, 1, 0, 0}
{0, 1, 1, 1, 1, 1, 0}
{0, 1, 1, 0, 0, 0, 1}
{0, 1, 1, 0, 0, 1, 1}
{0, 1, 1, 1, 0, 0, 1}
{0, 1, 1, 1, 0, 1, 1}
{0, 1, 1, 0, 0, 0, 0}
{0, 1, 1, 0, 0, 1, 0}
{0, 1, 1, 1, 0, 0, 0}
{0, 1, 1, 1, 0, 1, 0}
{0, 1, 0}
{0, 1, 0, 0, 1}
{0, 1, 0, 1, 1}
{0, 1, 0, 0, 0}
{0, 1, 0, 1, 0}
{0, 1, 0, 0, 1, 0, 1}
{0, 1, 0, 0, 1, 1, 1}
{0, 1, 0, 1, 1, 0, 1}
{0, 1, 0, 1, 1, 1, 1}
{0, 1, 0, 0, 1, 0, 0}
{0, 1, 0, 0, 1, 1, 0}
{0, 1, 0, 1, 1, 0, 0}
{0, 1, 0, 1, 1, 1, 0}
{0, 1, 0, 0, 0, 0, 1}
{0, 1, 0, 0, 0, 1, 1}
{0, 1, 0, 1, 0, 0, 1}
{0, 1, 0, 1, 0, 1, 1}
{0, 1, 0, 0, 0, 0, 0}
{0, 1, 0, 0, 0, 1, 0}
{0, 1, 0, 1, 0, 0, 0}
{0, 1, 0, 1, 0, 1, 0}
{0, 0, 1, 0, 1, 0, 1}
{0, 0, 1, 0, 1, 1, 1}
{0, 0, 1, 1, 1, 0, 1}
{0, 0, 1, 1, 1, 1, 1}

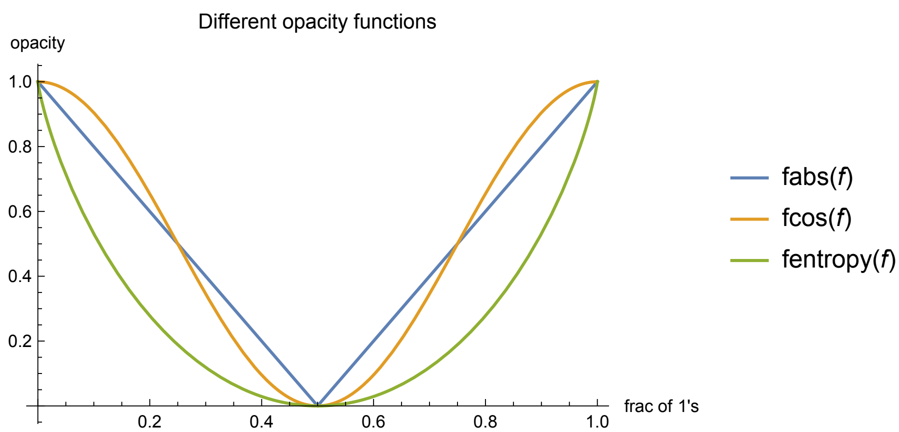



## Set opacity as function of “how close is 1/0 fraction to 50%?”

Now we decide how opaque to make a square, as a function of its fraction of 1’s and 0’s. The idea is that an equal proportion of 1’s and 0’s should result in an clear square (Opacity 0), whereas a bitstring with all 1’s or all 0’s should be totally opaque (Opacity 1). This amounts to choosing a function that takes the bitstring, computes the fraction of 1’s, and is 0 at 0.5 (50%) and 1 at 0 (0%) and 1 (100%).

In[29]:=
```
fabs[frac_] := 2 * Abs[frac - .5]
fcos[frac_] := 1/2 (Cos[2 π frac] + 1)
fentropy[frac_] :=
 If[frac == 0 || frac == 1, 1, 1 + (frac * Log2[frac] + (1 - frac) * Log2[1 - frac])]
Plot[
 {fabs[f], fcos[f], fentropy[f]}, {f, 0, 1},
 PlotLegends → "Expressions",
 PlotLabel → "Different opacity functions",
 AxesLabel → {"frac of 1's", "opacity"}
]
```

Out[32]=

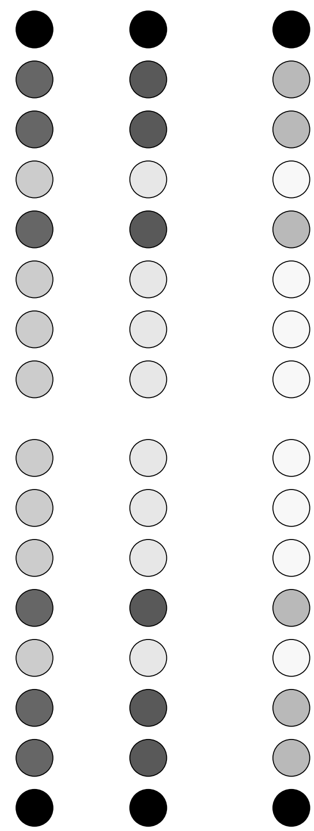



You can see that although the functions agree at 0, 0.5, and 1.0, they vary a little in their gray-levels in between:

In[33]:=
```
tab = Table[{b,
    Sequence @@ Table[
      Graphics[{
        EdgeForm[Black], Opacity[f[Total@b / Length@b]], Disk[]
       }, ImageSize → 20],
      {f, {fabs, fcos, fentropy}}]
   },
   {b, IntegerDigits[Range[0, 2^5 - 1], 2, 5]}];
TableForm[
 Join[tab[[ ;; 8]], {"..."}, tab[[-8 ;;]]],
 TableHeadings → {None, {"bitstring", "fabs", "fcos", "fentropy"}},
 TableDepth → 2,
 TableAlignments → Center
]
```

Out[34]//TableForm=

| bitstring | fabs | fcos | fentropy |
|---|---|---|---|
| {0, 0, 0, 0, 0} | | | |
| {0, 0, 0, 0, 1} | | | |
| {0, 0, 0, 1, 0} | | | |
| {0, 0, 0, 1, 1} | | | |
| {0, 0, 1, 0, 0} | | | |
| {0, 0, 1, 0, 1} | | | |
| {0, 0, 1, 1, 0} | | | |
| {0, 0, 1, 1, 1} | | | |
| ... | | | |
| {1, 1, 0, 0, 0} | | | |
| {1, 1, 0, 0, 1} | | | |
| {1, 1, 0, 1, 0} | | | |
| {1, 1, 0, 1, 1} | | | |
| {1, 1, 1, 0, 0} | | | |
| {1, 1, 1, 0, 1} | | | |
| {1, 1, 1, 1, 0} | | | |
| {1, 1, 1, 1, 1} | | | |



Fabs is a little darker, which makes the final image look more dramatic. But Shannon's entropy is more thematically fitting.

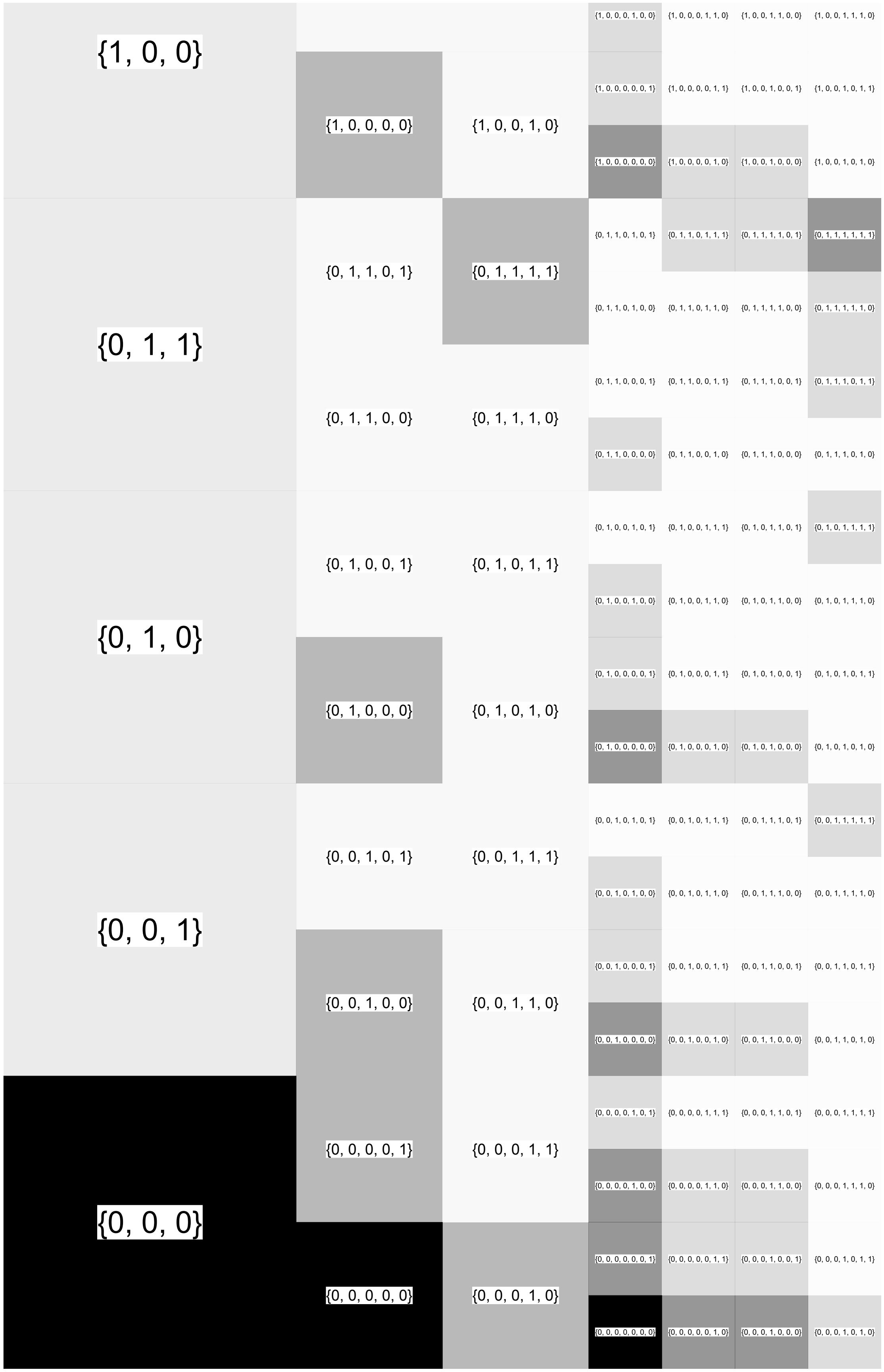
In[35]:=
```
opac[bits_] := fentropy[Total@bits / Length@bits]
```

Override the graphics function to use opacity instead of the parent's color:

```
In[36]:= square /: graphics[sq_square] := Graphics[{
    {
     Opacity[opac[bits[sq]]],
     Rectangle[pos[sq], pos[sq] + sidelen[sq] * {1, 1}]
    },
    Text[Style[bits[sq], FontSize → 20 * sidelen[sq]],
     pos[sq] + sidelen[sq]/2 * {1, 1}, Background → White]
   }]
```

```
In[37]:= Show[graphics[allSquares], ImageSize → 600]
```

Out[37]=


{1, 0, 0}
{1, 0, 0, 0, 0}
{1, 0, 0, 1, 0}
{1, 0, 0, 0, 1, 0, 0}
{1, 0, 0, 0, 1, 1, 0}
{1, 0, 0, 1, 1, 0, 0}
{1, 0, 0, 1, 1, 1, 0}
{1, 0, 0, 0, 0, 0, 1}
{1, 0, 0, 0, 0, 1, 1}
{1, 0, 0, 1, 0, 0, 1}
{1, 0, 0, 1, 0, 1, 1}
{1, 0, 0, 0, 0, 0, 0}
{1, 0, 0, 0, 0, 1, 0}
{1, 0, 0, 1, 0, 0, 0}
{1, 0, 0, 1, 0, 1, 0}
{0, 1, 1}
{0, 1, 1, 0, 1}
{0, 1, 1, 1, 1}
{0, 1, 1, 0, 0}
{0, 1, 1, 1, 0}
{0, 1, 1, 0, 1, 0, 1}
{0, 1, 1, 0, 1, 1, 1}
{0, 1, 1, 1, 1, 0, 1}
{0, 1, 1, 1, 1, 1, 1}
{0, 1, 1, 0, 1, 0, 0}
{0, 1, 1, 0, 1, 1, 0}
{0, 1, 1, 1, 1, 0, 0}
{0, 1, 1, 1, 1, 1, 0}
{0, 1, 1, 0, 0, 0, 1}
{0, 1, 1, 0, 0, 1, 1}
{0, 1, 1, 1, 0, 0, 1}
{0, 1, 1, 1, 0, 1, 1}
{0, 1, 1, 0, 0, 0, 0}
{0, 1, 1, 0, 0, 1, 0}
{0, 1, 1, 1, 0, 0, 0}
{0, 1, 1, 1, 0, 1, 0}
{0, 1, 0}
{0, 1, 0, 0, 1}
{0, 1, 0, 1, 1}
{0, 1, 0, 0, 0}
{0, 1, 0, 1, 0}
{0, 1, 0, 0, 1, 0, 1}
{0, 1, 0, 0, 1, 1, 1}
{0, 1, 0, 1, 1, 0, 1}
{0, 1, 0, 1, 1, 1, 1}
{0, 1, 0, 0, 1, 0, 0}
{0, 1, 0, 0, 1, 1, 0}
{0, 1, 0, 1, 1, 0, 0}
{0, 1, 0, 1, 1, 1, 0}
{0, 1, 0, 0, 0, 0, 1}
{0, 1, 0, 0, 0, 1, 1}
{0, 1, 0, 1, 0, 0, 1}
{0, 1, 0, 1, 0, 1, 1}
{0, 1, 0, 0, 0, 0, 0}
{0, 1, 0, 0, 0, 1, 0}
{0, 1, 0, 1, 0, 0, 0}
{0, 1, 0, 1, 0, 1, 0}
{0, 0, 1}
{0, 0, 1, 0, 1}
{0, 0, 1, 1, 1}
{0, 0, 1, 0, 0}
{0, 0, 1, 1, 0}
{0, 0, 1, 0, 1, 0, 1}
{0, 0, 1, 0, 1, 1, 1}
{0, 0, 1, 1, 1, 0, 1}
{0, 0, 1, 1, 1, 1, 1}
{0, 0, 1, 0, 1, 0, 0}
{0, 0, 1, 0, 1, 1, 0}
{0, 0, 1, 1, 1, 0, 0}
{0, 0, 1, 1, 1, 1, 0}
{0, 0, 1, 0, 0, 0, 1}
{0, 0, 1, 0, 0, 1, 1}
{0, 0, 1, 1, 0, 0, 1}
{0, 0, 1, 1, 0, 1, 1}
{0, 0, 1, 0, 0, 0, 0}
{0, 0, 1, 0, 0, 1, 0}
{0, 0, 1, 1, 0, 0, 0}
{0, 0, 1, 1, 0, 1, 0}
{0, 0, 0}
{0, 0, 0, 0, 1}
{0, 0, 0, 1, 1}
{0, 0, 0, 0, 0}
{0, 0, 0, 1, 0}
{0, 0, 0, 0, 1, 0, 1}
{0, 0, 0, 0, 1, 1, 1}
{0, 0, 0, 1, 1, 0, 1}
{0, 0, 0, 1, 1, 1, 1}
{0, 0, 0, 0, 1, 0, 0}
{0, 0, 0, 0, 1, 1, 0}
{0, 0, 0, 1, 1, 0, 0}
{0, 0, 0, 1, 1, 1, 0}
{0, 0, 0, 0, 0, 0, 1}
{0, 0, 0, 0, 0, 1, 1}
{0, 0, 0, 1, 0, 0, 1}
{0, 0, 0, 1, 0, 1, 1}
{0, 0, 0, 0, 0, 0, 0}
{0, 0, 0, 0, 0, 1, 0}
{0, 0, 0, 1, 0, 0, 0}
{0, 0, 0, 1, 0, 1, 0}

## Use as image mask

In[38]:=
```
im = Import["IMG_0795-001.JPG"];
Thumbnail[im, 500]
```

Out[39]=

In[40]:=
```
imrot = ImageRotate[im, Top → Left];
```

Turn off the text of the bitstring:

In[41]:=
```
square /: graphics[sq_square] := Graphics[{
    Opacity[opac[bits[sq]]],
    Rectangle[pos[sq], pos[sq] + sidelen[sq] * {1, 1}]
   }]
```

In[42]:=
```
allSquares = NestList[makeChildren, initials, 6];
```

There are a lot of squares. Each square turns into 4 squares in the next “generation”, starting with 8 squares and lasting 7 generations.

In[43]:=
```
allSquares // Flatten // Length
```

Out[43]= 43 688

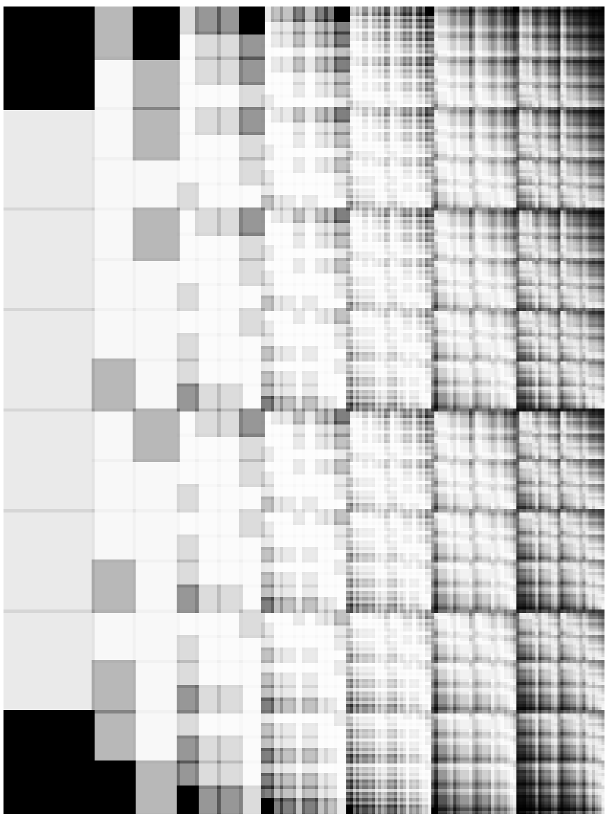
In[44]:=
```
Clear[x];
x[1] = 8;
x[i_] := 4 * x[i - 1];
```
$$\sum_{i=1}^{7} x[i]$$

Out[47]= 43 688

In[48]:=
```
overlay =
  Show[graphics[allSquares], Frame → False, AspectRatio → ImageAspectRatio[imrot]];
```

In[49]:=
```
Rasterize[overlay, RasterSize → 200]
```

Out[49]=



In[50]:=
```
imfinal = ImageCompose[imrot, overlay] // ImageRotate[#, Top → Right] &;
Thumbnail[imfinal, 500]
```

Out[51]=

In[52]:=
```
Export["art-csep-pearson-FROM-MMA-vEntropy-INTERMEDIATE.jpg",
 imfinal, "CompressionLevel" → 0]
```

Out[52]= art-csep-pearson-FROM-MMA-vEntropy-INTERMEDIATE.jpg

(Now add the numbers manually (I did it in Mac's Preview.))