Formally describe an artwork

Choose one Sol LeWitt artwork. Unlike our previous activity at WMCA, this time, you should choose an artwork whose component elements are things you think that you can describe. As before, we will limit ourselves to a "mechanical description" of each artwork.

Do the following:

1. Snap a photo with your smartphone.
2. Try to determine the set of words that can be used to describe the artwork, drawn from two categories:
   (a) primitives, and
   (b) combining forms.

   Be sure to define the words you use before you use them. Try to be precise; if you can be mathematical in your precision, even better, although this is not strictly necessary.
3. Describe the artwork as completely as you can. If you find that your set of words is missing something, or if another word could be used to better describe the artwork, go back to step 2 and modify your set of words.

Because you might be wondering what is meant by primitive and combining form, here are some definitions.

A primitive is a type of data drawn from a (possibly infinite) set. It is atomic in the sense that it has no obvious constituent parts, or can be defined in a way that it has no constituent parts. For example, an `integer` is often taken to be a primitive in a programming language. Does it have constituent parts? Well, yes, in a way– from the machine's vantage point it can see and access an `integer`'s individual bits. However, from the vantage point of a programming language (like Java), it is often taken to be indivisible.

A combining form is a language element that describes some kind of composition. In typical programming languages, we often have two kinds of combining forms: those for data, and those for functions. For example, a combining form for data might be a `struct` in C, a `class` in Java, or a `type` in F#. A `class`, for instance, allows a user to combine multiple pieces of data (i.e., the class's `fields`) into a single piece of data. Many combining forms are also recursive in the sense that if primitives are data, and a combining form for data is data, then combining forms can combine other combining forms. For example, classes can have classes as their fields. A combining form for functions in all of these languages is a function definition. As you are well aware, functions can be built from other functions.

As an example for this exercise, suppose that we want to describe a line. Is a line a primitive or a combining form? It depends on how we want to describe it. Suppose we decide that a line is going to be a combining form. How can we define it? Here's one way.

Let `number` be a real number from $-\infty$ to $+\infty$, represented by the F# data type:
`type number = float`.

Let `coordinate` be a combining form consisting of a pair of `number`s, represented by the F# type:
`type coordinate = x:  number * y:  number`.

Let `line` be a combining form consisting of a pair of `coordinate`s, represented by the F# type:
`type line = start:  coordinate * end:  coordinate`.

At some point, all of thßese might be tied together using something like an `canvas` type:

```
type canvas =
| lines of line list
| // other things that can be on the canvas
```

For now, let's focus on defining the words to describe what we see. Once you have what you think comes close to defining all of the pieces of an artwork, try writing a sample program. Feel free to invent syntax represented by your data type. For example,

```
line starting at 3,4 and ending at 5,5
line starting at 1,1 and ending at 9,2
line starting at 3,4 and ending a 20,1
```