Learn Structure Synth ： Structure Synth を学ぶ

States, transformations and actions. ： 状態と変換と動作

Structure Synth is all about *states*. A state describes the current coordinate system and the current coloring mode. The coordinate system determines the position, orientation and size of all object drawn while in the current state.

Stucture Synth では形の「状態」を制御することが動作のすべてです。この状態と言うのは、対象とする物体の現在の座標軸や色設定の情報を記述しています。この物体を規定する座標軸では、現在の状態でのすべての物体の位置や方向や規模を決定します。

States are modified by *transformations*. For instance we can move the coordinate system one unit in the x-direction by applying the transformation: `{ x 1 }`. Similarly we can rotate the coordinate system 90 degrees about the x-axis by applying:`{ rx 90 }`. States are automatically combined while parsing, that is `{ x 1 x 1 }` is equal to `{ x 2 }`.

状態は「変換」によって変化してゆきます。たとえば、X方向に向けて1単位長さ分だけ座標軸を移動させるためには、変換 { x 1 } を適用することによって可能です。同様に、X軸回りに関して90度だけ座用軸を回転させるためには、{ rx 90 } を適用することによって可能です。この状態は、記述を解釈する過程において自動的に合成されますので、たとえば { x 1 x 1 } は、{ x 2 } と同じことになります。

States can be combined with *rule calls* to create *actions*. `{ x 2 } box` is an example of a transformation followed by a rule call. 'box' is a built-in rule. Not surprisingly, this rule draws a box located at (0,0,0) -> (1,1,1) in the current coordinate system.

状態は、動作を生成するために規則の名称と合成することができます。たとえば、{ x 2 } box は、次に示すような規則を呼び出すことで変換を行う1つの例になります。ここで ‘box’は、組み込まれた規則です。予想している通り、この規則では、現在の座標系において対角線が（0,0,0）→（1,1,1）になる位置に立方体を描画します。

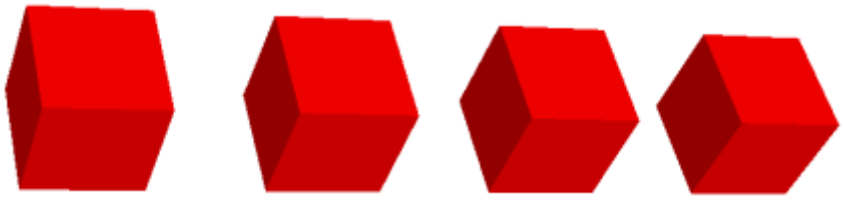
Now take a look at the following example:

それでは、以下の例題を見てみましょう。

```
box
{ x 2 } box
{ x 4 } box
{ x 6 } box
```

This creates the following output:

これより、以下に示す結果が得られます。



This produces four boxes with equal space between them. Notice that when we have multiple actions following each other like this they are all applied to the same state - in this case the initial state. The actions are **not** applied to the state produced by the previous action - this would have create an uneven spacing.

この動作により、4つの立方体がそれぞれ等しい間隔をもって描画されます。注意として、ここで示したような複数の動作がそれぞれ指示されるときには、これらはすべて同じ状態ーこの場合には初期の状態ーに対して適用されます。言い換えれば、この動作は前の動作によって生成された状態に対して適用されるわけではありません。もしこのようになるならば、等しくない間隔が生成されることになるはずです。

Iterated actions ： 繰り返す動作

It is possible to apply *iterated* actions, this is done using the multiplication symbol: for instance `3 * { x 2 } box` would be equal to creating three actions:

動作の記述においては、繰り返す動作を適用することができます。これは、掛け算の記号 ＊ を用いて指定します。たとえば、 3 * { x 2 } box は、以下の3つの動作が生成したものと同じようになります。この繰り返す場合の変換 x は、直前の状態に対して行われます。

```
{ x 2 } box
{ x 4 } box
{ x 6 } box
```

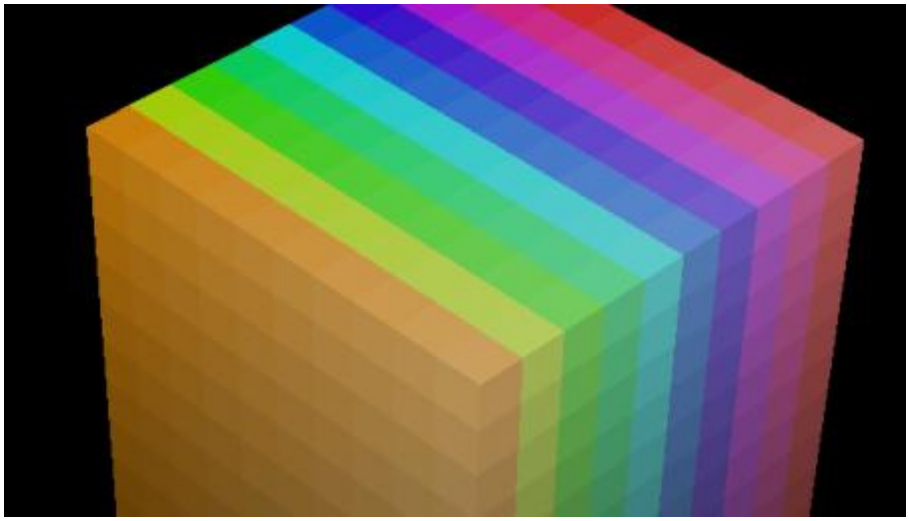
Color transformations ： 色設定の変換

Similar to the spatial transformations it is also possible to transform the current rendering color. Structure Synth uses HSV (Hue, Saturation and Value) for representing colors - this is perhaps not as familiar as the RGB color model, but offers a slightly more intuitive representation once you get used to it (at least that is what some people claim - personally I still find it easier think in terms of red, green and blue components). The color transformations are applied using the 'hue', 'saturation' and 'value' operators.

同様な変換のなかで特別なものとして、現在の色情報を変換することもできます。Structure Synth では、物体の色設定において、HSV（色相・彩度・明度）を用いています。たぶんこの設定方法は、RGB の色設定のようには慣れていないかもしれません。しかし、一度その方法に慣れてしまえば、多少は直観的な表現方法であることが理解できます。（とにかく、難しいと主張する人もいますが、個人的には、赤・緑・青の成分を構成して色を考えるよりは、楽に設定できると思います。） 色の変換は、色相 hue ・彩度 sat ・明度 b の命令を用いて設定できます。

The next example demonstrates both iterated actions and color transformations to draw a nice color cube:

次の例では、繰り返し動作と色の変換の両方を行い、きれいな色の立体を描画する例をしめしています。

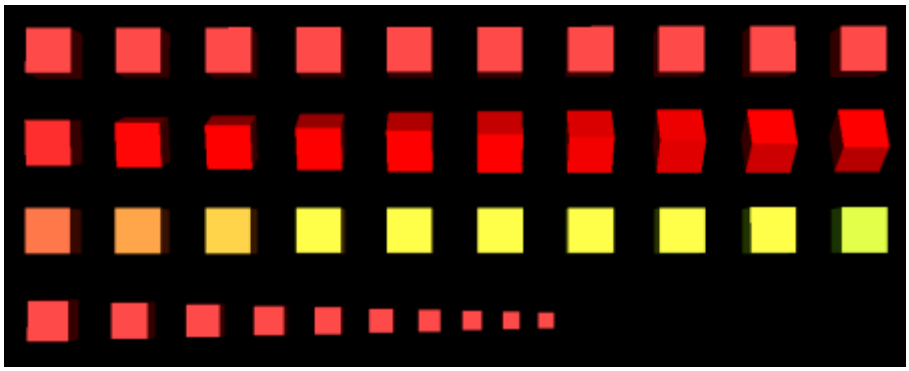


```
10 * { x 1 hue 36 } 10 * { y 1 sat 0.9 } 10 * { z 1 b 0.9 } box
```

ここで使われている変換は、hue は色相（色味の変化）、sat は彩度（色の強さ）、b は明度（色の明るさ）になります。

Here is another example demonstrating different kinds of transformations:

ここではもう一つの例として、異なる種類の様々な変換を示しています。



```
10 * { x 2 } box

1 * { y 2 } 10 * { x 2 rx 6 } box

1 * { y 4 } 10 * { x 2 hue 9 } box

1 * { y 6 } 10 * { x 2 s 0.9 } box
```

上記の例では変換として、x は物体の移動、rx は物体の回転、hue は色相の変化、s は物体の大きさになります。

Built-in rules ： 組み込まれた規則

The Box is an example of one the primitives - built-in rules - in Structure Synth. The other built-in rules are: Sphere, Dot, Grid, Line, Cylinder, Mesh,CylinderMesh.

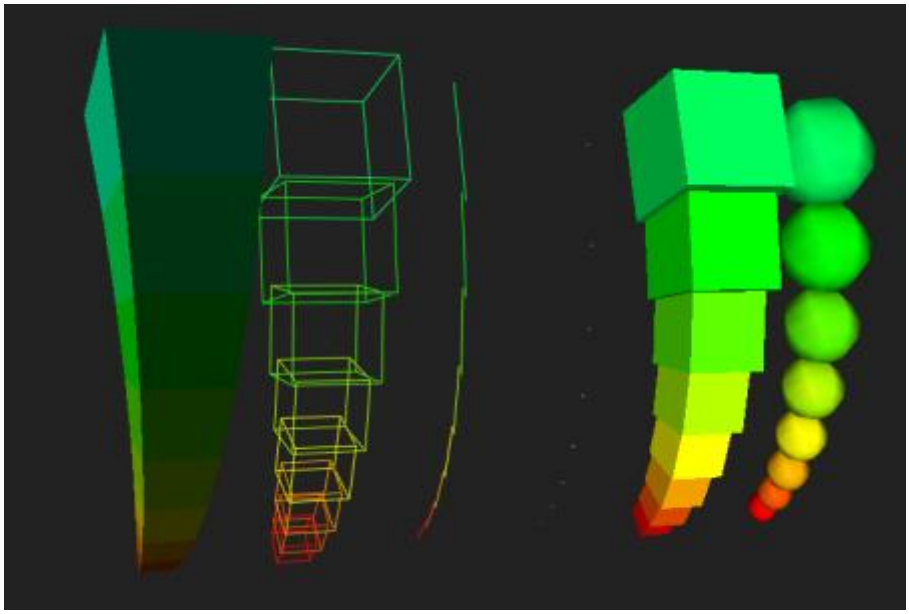
これまでの例で使われた box は、Structure Synth において組み込まれた規則として生成できる形状 (primitive) の中の1つで、直方体になります。その他の組み込まれた形状を生成する規則としては、球(sphere)、点(dot)、直方体の枠(grid)、線(line)、連続面(mesh) などがあります。

なお、cylinder, cylindermesh, triangle, tube などがありますが、

未実装であったり特殊な使い方になります。

This example demonstrates different primitives (from left to right: mesh, grid, line, dot, box, sphere):

以下の例では、いくつかの形状を示しています。(左から右に向かって、連続面、直方体の枠、線、点、直方体、球)



Making Rules ： 規則の作り方

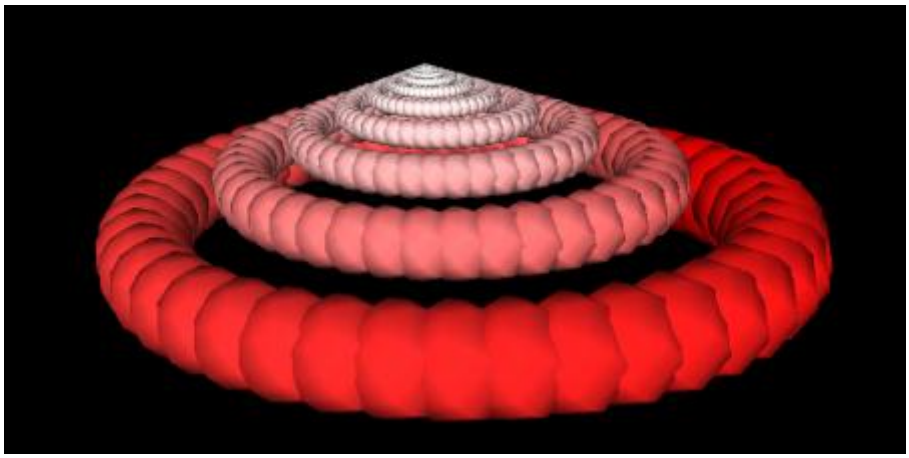
Custom rules are the key to creating complex and sophisticated structures. Rules are created using the 'rule' keyword. A rule can used the same way as any built-in primitive. The most important aspect of rules are, that they are able to call themselves. Take a look at the following example:

独自の規則は、複雑で洗練された構造体を生成するための鍵になります。規則は、'rule' コマンドを使って作られます。作られた規則は、組み込まれた形状と同じ方法で利用することが出来ます。規則の最も重要な特徴は、自分自身を呼び出すこと(再帰呼出)が出来ることです。以下の例で調べてみましょう。

```
R1


rule R1 {
```

```
  { x 0.9 rz 6 ry 6 s 0.99  sat 0.99  } R1
  { s 2 } sphere
}
```

Which generates this output. 

先の例から、上記の出力が生成されます。

Notice that this rule recursively calls itself. It would never terminate - however Structure Synth has a default maximum recursion depth of 1000 recursions. This value can be changes using the 'set maxdepth xxx' command. Another way to force termination would be using the 'set maxobjects xxx' keyword, which makes Structure Synth keep track of the number of objects drawn.

この規則では、自分自身を再帰的に呼び出していることに注意してください。このままでは決して終わらないように思えますが、実は Structure Synth では、標準の最大再帰呼出を 1000 回と定めています。この最大値は、'set maxdepth xxx' コマンドを用いて変更することができます。意図的に終わらせるためのもう1つの方法は、'set maxobjects xxx' コマンドを用いることで、Structure Synth は、生成される物体の個数を管理して制限することができます。

Adding some randomness ： 乱れを導入する

Now, in order to things interesting, we will probably want to create something less static - by adding some randomness. In Structure Synth this is achieved by creating multiple definitions for the same rule:

さて、面白い表現を得るためには、固定的に生成されるものより、多分いくらかの乱れを導入することが必要です。Structure Synth では、同じ名称の規則において、異なる複数の定義を導入することで実現しています。

```
R1


rule R1 {
```

```
  { x 0.9 rz 6 ry 6 s 0.99  sat 0.99  } R1
  { s 2 } sphere
}

rule R1  {
  { x 0.9 rz -6 ry 6 s 0.99  sat 0.99  } R1
  { s 2 } sphere
}
```

Notice the 'R1' rule has two definitions. Now, whenever the Structure Synth builder needs to call the 'R1' rule, it will choose one of the definitions at random, resulting in something like the following image:

この例では、'R1' の規則が2つ定義されています。こうすると、Structure Synth が 'R1' 規則を呼び出す必要があるときには、複数の定義の中の1つをランダムに選択することになります。この結果、以下の様な画像が生成されます。