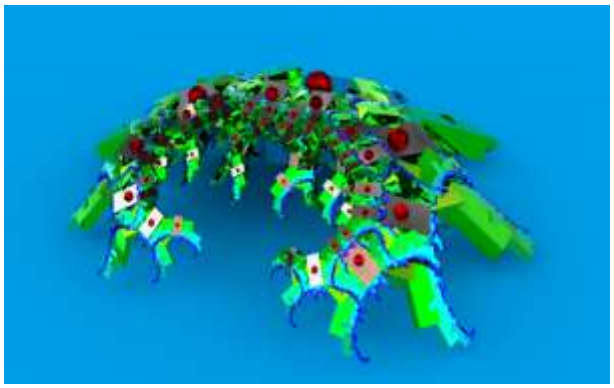
Structure Synth Tutorial ： StructureSynthのチュートリアル

2011-05-16

I love Structure Synth because it is a quick and easy way to generate 3D artwork. It's very programmer oriented. Rather than using the mouse to position primitives, you position them using a language called EisenScript. First Download Structure Synthand install it before continuing with this tutorial. Structure Synth is developed byMikael Hvidtfeldt Christensen and available for Windows, Mac, and Linux.

私は Structure Synth が気に入っています。なぜなら、3次元モデルとしてのアートを簡単に素早く作り上げる方法だからです。でもそれは、プログラマーに限ってのことです。なぜなら、物体の位置はマウスで指定するのではなく、EisenScript と呼ばれるプログラム言語を用いて、物体の位置を決めるからです。それでは、Structure Synth をダウンロードしてインストールしてから、このチュートリアルを始めましょう。なお、Sturcture Synth は、Mikael Hvidtfeldt Christensen によって開発され、Windows だけでなく Mac や Linux でも動作します。



This "Fractal Crab" is an example of a Structure Synth creation using recursion, rendered with Sunflow. The full sized 2560x1600 version is here.

この作品「Fractal Crab」は、再帰的呼出を用いて Structure Synth により作られたもので、Sunflow を用いて3次元CGの画像を作っています。なお、この画像の元ファイル 2560x1600 版は、ここにあります。

A Simple Scene　単純な作品

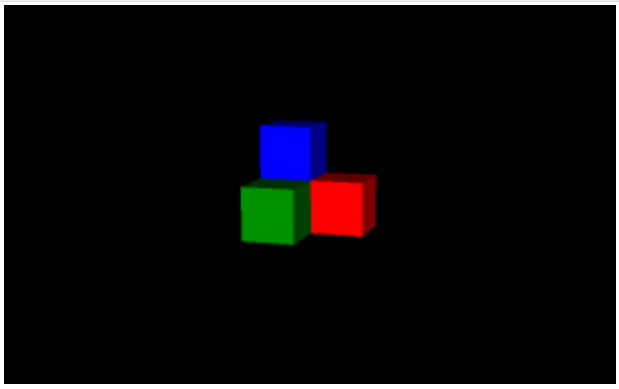
The first scene creates three boxes, each offset in the x, y, or z direction. Click and drag in the view to rotate the scene. Shift-drag※ to zoom the scene in and out, and hold down the command(windows) key to translate the scene horizontally or vertically.　※click の誤記を訂正

最初の作品では、3つの箱を作ります。それぞれがX、Y，Zの方向にずれています。表示の中でクリックやドラッグをして物体を回すことができます。またシフト＋ドラッグで物体に近づいたり遠ざけたりできます。さらに、windows キーを押しながらドラッグすると、作品を水平や垂直に平行移動が可能です。

By rotating the scene slightly you can see that the green box, offset 1 in the z direction, comes out of the screen towards you. The x and y offset are as you'd expect, to the right, and up. If you get lost, click the "Reset View" button to return the default view, looking towards {0,0,0} down the z axis.

作品をわずかに回転させることにより、緑色の箱が原点からZ軸方向に1だけずれていることが分かります。これは、画面からこちらに向かってくる方向になります。見て分かる通り、X軸とY軸のずれは、右側と上側になります。もし視点を見失った場合には、ツールの上部にある「Reset View」ボタンを押すことで、初期状態に戻ることが出来ます。この状態は、Z軸上から原点{0,0,0}を見ることになります。

```
{x 1 color red} box
{y 1 color blue} box
{z 1 color green} box
```



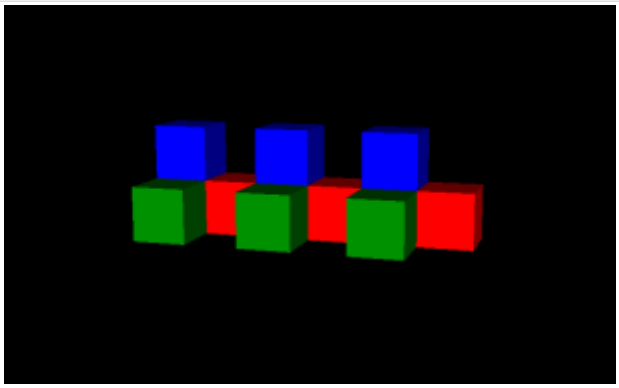
Now that you have your bearings in 3D space, the next step is to put our creation into a rule so we can reuse it. This scene uses the 'threeBox' rule several times, offsetting it 2 units each time.

それでは、3次元空間での物体の生成を始めましょう。次のステップでは、後から再利用できるように、独自の規則を作ることを試してみます。この作品では、「threeBox」と言う規則が、それぞれX軸方向に2単位づつずれて、何回か使われています。

```
{x 0} threeBox
{x 2} threeBox
{x 4} threeBox

rule threeBox
```

```
{
    {x 1 color red} box
    {y 1 color blue} box
    {z 1 color green} box
}
```

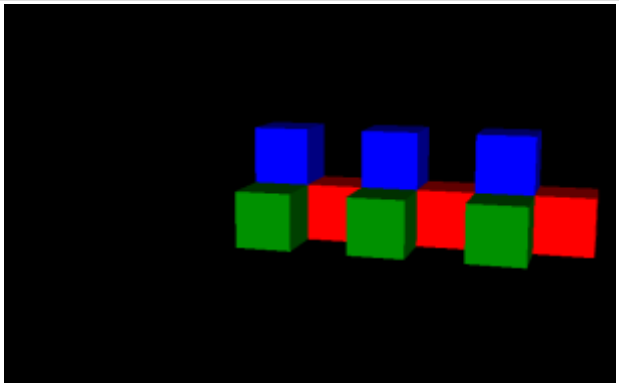


This works, but it's not how a programmer would write it. We can refactor it like this. This is evaluated something like "Translate 2 in the x direction, three times, and call threeBox each time".

これはこれで動作するのですが、しかしプログラマーはこのようには記述しません。むしろ、以下の様に書き直すでしょう。この例では、X軸方向に2だけ平行移動することを3回繰り返し、その都度、threeBox を呼び出しています。

```
3 * {x 2} threeBox

rule threeBox
{
    {x 1 color red} box
    {y 1 color blue} box
    {z 1 color green} box
}
```



Unfortunately this gives a slightly different result than the previous scene. Here's how to shift the first threeBox to the left of the origin, before repeating it three times.

同じようなスクリプトの記述ですが、先の作品と比べて、若干異なった結果になってしまいます。なぜなら、threeBox が3回繰り返される前に、原点から左に移動してから呼び出されているからです。

```
1 * {x -2} 3 * {x 2} threeBox

rule threeBox
{
    {x 1 color red} box
    {y 1 color blue} box
    {z 1 color green} box
}
```

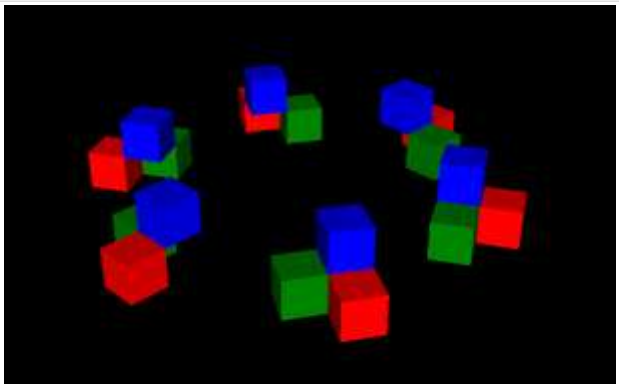
Rotation　回転

We can combine repeating, rotation, and offset to make rings. The '6 * {ry 60}' repeats the threeBox rule 6 times, rotating it 60 degrees each time. Then, the '1 * {x 4}' translates it 4 units from the origin in the direction of the rotation.

ここでは、繰り返しと座標回転と増分移動を組み合わせることで、環状の作品を作る方法を示します。まず「6 * {ry 60}」は、threeBox 規則を6回呼び出すときに、各回で座標軸をY軸周りに60度ずつ増分させることです。次に「1 * {x 4}」は、回転する軸から半径方向に4単位長さだけ平行移動させています。

```
6 * {ry 60} 1 * {x 4}  threeBox

rule threeBox
{
    {x 1 color red} box
    {y 1 color blue} box
    {z 1 color green} box
}
```



Try changing the 6, 60, 1, and 4 values on the first line and observe how each alters the appearance of the scene.

それでは、上記のスクリプトの1行目にある6、60、1、4などの値を変化させて試してみましょう。そこでは、作品の表現において、それぞれの値の変化が、どのように影響するか観察してみましょう。

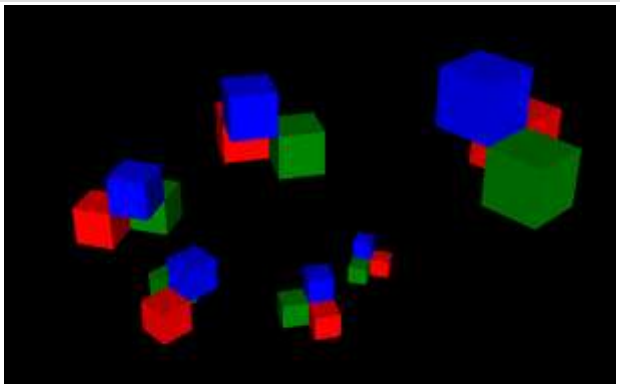
Scaling　規模の変化

Now lets try scaling our boxes each time we repeat by scaling them to 0.75 each time we repeat them. The example below adds a 's 0.75' to the first repeated block.

それでは次は、先に作っていた3つの箱を、生成するたびに規模（大きさ）を変化させてみます。具体的には、生成を繰り返すときに 0.75 倍として（縮小させて）います。以下の例では、先に用いたスクリプトにおいて、最初に生成する物体に対して「s 0.75」を加えることになります。

```
6 * {ry 60 s 0.75} 1 * {x 4}  threeBox

rule threeBox
{
    {x 1 color red} box
    {y 1 color blue} box
    {z 1 color green} box
}
```



Recursion　再帰的呼出

The script below is an example of a recursive rule. You'll notice there is an 'md 8' where we defined this rule. This is short for saying 'maxdepth 8'. Both 'maxdepth' and 'md' are interchangeable. You'll notice recursiveRule calls itself after making a box. Also new in this example is a 'hue 60' declaration. This shifts the hue 60 degrees on the color wheel each time the rule is evaluated.

以下に示すスクリプトは、規則の再帰的呼出の例になっています。ここでは、規則を定義するところで、「md 8」があることに注意してください。これは、「maxdepth 8」の省略の記述です。この「maxdepth」と「md」はどちらも同じ意味になります。さて本題ですが、box を生成した後で、recursiveRule の定義

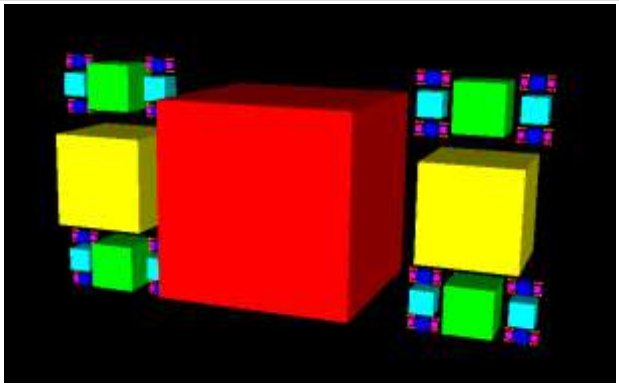
の中で自分自身を呼び出しています。これが再帰的呼出です。またこの例での新しい機能としては、「hue 60」の設定があります。これは、規則が呼出され実行するときに、その都度、色空間上の色相を 60 度ずつ変化させています。

Try adjusting the rotation angles, axes, and scaling to see what effect each has on this scene.

この作品では、回転角度の調整、平行移動、規模の変化、などがどのように影響しているか、確認することが出来ます。

```
recursiveRule

rule recursiveRule md 8
{
    box
    {x 1 rz 90 s 0.5 hue 60} recursiveRule
    {x -1 rz 90 s 0.5 hue 60} recursiveRule
}
```



Randomness 乱数的実行

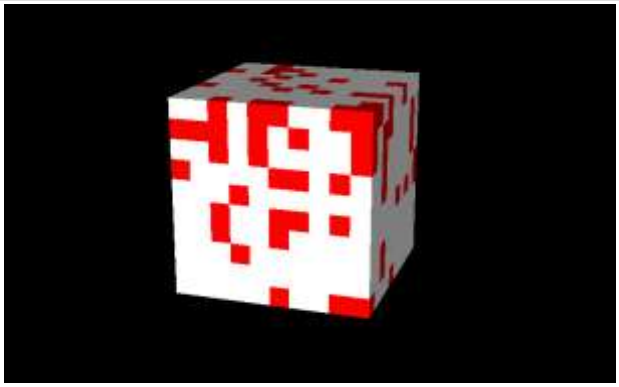
Randomness is implemented in Structure Synth by implementing two or more versions of a rule with the same name. Structure Synth will randomly choose one or another based on the weights you specify. In the example below, the red box rule's weight is reduced so it is called less by adding 'w 0.25'. You can also increase the rule's weight by choosing a value greater than 1.

Structure Synth において乱数的実行は、同じ名称で複数の異なる内容の規則を定義することにより実現されます。Structure Synth は、規則に指定された重み付けに従って、複数の中から1つの規則を乱数的に選択します。以下の例では、赤い箱を生成する規則は、重み付けとして「w 0.25」と付け加えることにより、呼び出されることが少なくなります。同様に、規則の重み付けを1よりも大きな値を選択すると、その規則の呼出が多くなります。

```
10 * {x 1} 10 * {y 1} 10 * {z 1} oneBox

rule oneBox w 0.25
{
    {color red} box
}

rule oneBox
{
    {color white} box
}
```



Combining Randomness and Recursion　乱数的実行と再帰呼出の複合

This example will make a branching tree. Click the blue arrow to regenerate the scene and see a different random tree. Each time you click the blue arrow, it will increment the seed. If you saw a tree you liked, you can decrement the seed and regenerate the same tree.

この例では、枝分かれした樹状物を作成しています。ツール上部の青丸矢印ボタンをクリックするために、新しい形状が生成され、毎回異なって樹形が見られます。ここでツールのボタンをクリックするたびに、乱数の初期値(seed)が1づつ増加しています。もし、気に入った樹形が生成されたときは、この乱数の初期値を調整して、改めて形状を生成すれば、気に入った樹形を再現することが出来ます。

This example defines a branch rule. Every time branch calls itself, it reduces the size of the subsequent block by a bit, and translates the box higher. Every now and then the tree will 'fork' and call the rule that results in two branches at a node.

この例では、枝分かれの規則を定義しています。再帰的に規則 branch が呼出されるたびに、続いてゆく枝の大きさが少しずつ小さくなってゆき、さらに box は高い位置に平行移動してゆきます。また樹形の成長において、時々ですが枝分かれ fork してゆきます。これは、枝分かれする節において、2つの branch を呼び出す規則が乱数的に使われた結果です。

```
branch

// Create a step or trunk that doesn't fork.
rule branch md 20 w 3
{
    {s 0.95 y 1  hue 20 ry 90} branch
    box
}

// "Fork" and create two branches.
rule branch md 12
{
    {s 0.85 y 1 rz 25 hue 20} branch
    {s 0.85 y 1 rz -25  hue 20} branch
    box
}
```

You can put a sphere on the end of each branch like this.

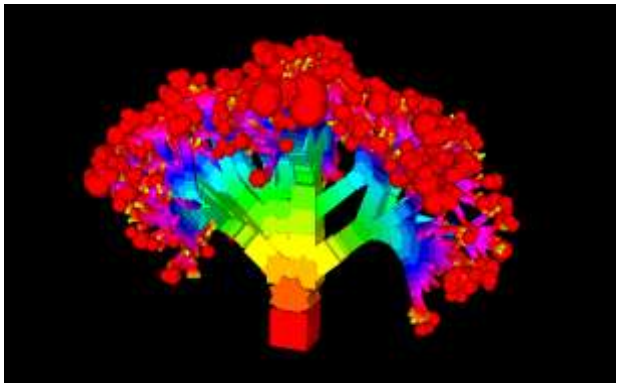
さらにこの例のように、各枝の先端において球形を加えるには、規則 branch において再帰的な呼出回数の制限 10 を超えたときに、規則 apple を呼び出すことで実現できます。

```
branch

rule apple
{
    {s 4 color red} sphere
}

// Create a step or trunk that doesn't fork.
rule branch md 10 > apple
{
    {s 0.95 y 1  hue 20 ry 90} branch
    box
}

// "Fork" and create two branches.
rule branch maxdepth 12
{
    {s 0.85 y 1 rz 25 hue 20} branch
    {s 0.85 y 1 rz -25  hue 20} branch
    box
}
```



The next step is rendering in a ray tracer! You can use any of the export templates provided to generate a scene file you can render in Sunflow or POVray. Rendering will be a topic for a future article. Have fun!

さてこの次の段階では、レイトレーシングを用いて綺麗な画像を生成することになります。この Structure Synth では、いくつかの外部利用のシーンデータを出力することができ、これにより Sunflow や POVray が利用できます。このようなレンダリングは、今後の話題としておきます。それでは、Structure Synth を楽しんでください。