Computer Science 112
Art & Science of Computer Graphics
The College of Saint Rose
Fall 2015

Topic Notes: Random Numbers

For every model we have seen so far, running the model repeatedly always gives exactly the same image, every time. But this does not need to be the case. Many computer programs work this way: they are entirely *deterministic*.

However, this does not need to be the case. We can also make parts of our programs *non-deterministic* – that is, they behave differently each time. This is accomplished through selection of *random numbers*. There are many places in computing where this is not just useful, but essential. When simulating a card game, we would want to shuffle the cards. In a game involving dice, we would want to roll the dice. Computer-controlled characters in games often exhibit some randomness in their behavior. Even in some scientific and engineering algorithms, where you would expect there to be a strict mathematical formula that determines our answers, introducing some randomness can help find possibly find solutions more quickly.

Like nearly all programming languages, Python provides us with mechanisms to generate random numbers. The most basic is the function `random`, which is part of standard Python. To use it at the `python3` command line:

```
>>> import random
>>> random.random()
0.45285062340015103
>>> random.random()
0.965792373456216
>>> random.random()
0.0697553071172271
>>> random.random()
0.7153488104638596
```

In order to use the `random.random()` function, we need to include a different `import` statement. Once we have done that, we can call `random.random()` to obtain random numbers. If you run these same statements, you'll certainly get different numbers, but they're all in the same range: between 0.0 and 1.0.

Sometimes, that's exactly what you want, but often it is more useful to get random values in some other range. Python can help us here as well. The function `random.uniform()` is given two numbers: the upper and lower bounds of the range of possible numbers we want to use. For example:

```
>>> random.uniform(0,10)
```

```
4.257705194235514
>>> random.uniform(0,100)
1.3934249989294378
>>> random.uniform(50,100)
64.2784128916245
>>> random.uniform(-50,50)
-27.612715670594923
>>> random.uniform(0.1,0.2)
0.11172933166957517
```

With `random.uniform()`, even when we specify whole numbers (*integers*) for the bounds of our random range, we get numbers with fractional parts (digits after the decimal point). In math, these would be called *real numbers*, but in computing, these are normally called *floating-point numbers* because of the way they are stored inside the computer. We will use the terms "integers" and "floating-point numbers".

In some circumstances, we will need or prefer to have integer values for our random numbers. Python has us covered. A third function, `random.randint()` does just this:

```
>>> random.randint(0,10)
10
>>> random.randint(0,10)
4
>>> random.randint(0,10)
3
>>> random.randint(10,20)
14
>>> random.randint(10,20)
17
>>> random.randint(-5,5)
-4
```

Much like what we do for `random.uniform()`, we specify the smallest and largest values we are interested in having as possible values, and `random.randint()` will choose a number for us within that range. Note that the range here is inclusive – both the upper and lower bound values are among the possible random numbers generated.

This is enough to get us started on adding some non-determinism to our models. We can generate and use these random numbers anywhere we'd like to replace some fixed value with something random.

Our first example with random numbers will simply choose some random positions for objects in our scene:

On the Wiki: Randomconeplacement

Making things a bit more random, we can choose the number of cones randomly within a range. Note that this needs to be an integer value, so we use `randint` instead of `uniform`.

On the Wiki: Randomconecount

Now, random heights!

On the Wiki: Randomconeheights

And totally randomly-colored plaster materials!

On the Wiki: Randomconecolors

Note that we finally found a use for `random.random()`, since we just happen to need random values in the 0.0-1.0 range.

Now instead of totally random colors, we will use HSV to have a random hue but always 1 for saturation and value to get the most bright and intense colors for our plaster materials.

On the Wiki: Randomconecolors2

And for a different example that uses programmed repetition with some randomization, the salt coming out of a salt shaker:

On the Wiki: Saltshaker

Note that here we are randomly choosing rotation amounts for the orientation of each salt grain, then randomly rotating about the y-axis after translating to wind up with a cylindrical column of salt grains.