The test world.

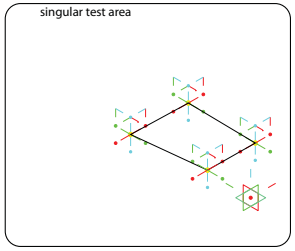
Usually while we are using GC we have pretty complicated ideas.
‘I’d like to make a component that will be applies to 30 000 polygons, and react to 5 different possible input conditions in a totally smooth way’
At best that’d be a brave thing to jump right into, usually, it’d be prohibitively confusing. The best thing to do is to make a test world that isolates the component and allows us to put it through its paces in a nice uncluttered way.

Without being explicitly aware of it, just making a component on a polygon that you are then going to applying to a grid of polygons is a test world. It tests to see if the component works or not, which is a pretty big thing really.

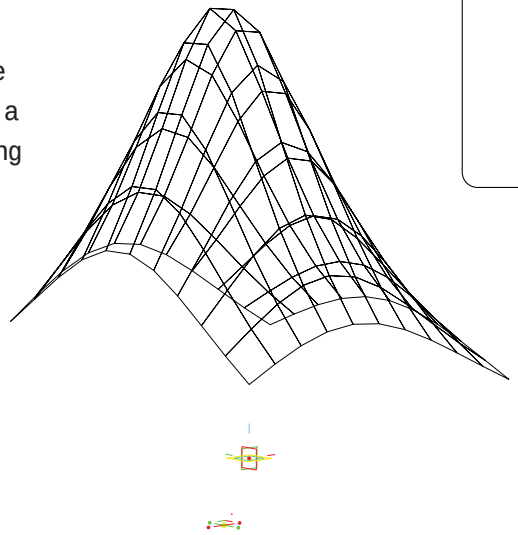
If our component is a bit more complicated, and has features that react to a few inputs in different ways then we need to be able to model these different inputs in the test world too.

Initially this seems just like making the process of building a component just sound a bit complicated, but it’s what happens next that’s the interesting bit.

Now we try and break the component!! Or at least make it behave in a way that we consider ‘wrong’

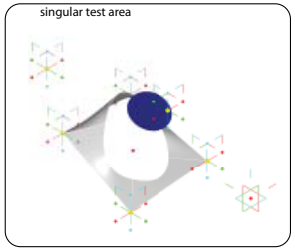


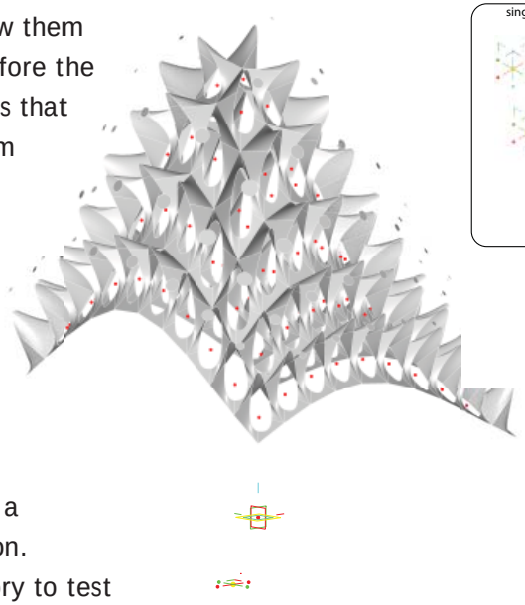
I’ve included a simple component that does a load of stuff depending on its relationship to a few points, and it’s orientation on a surface. At the moment you don’t need to be too concerned about how the internals of the component actually work, just that it does (although there are some fun tricks in there that you might find useful if you dig around a bit).



When I’m testing I use two levels of application. The base level where it’s just one polygon, and then I wiggle the points that control the corners about, and play with the variables and see if it works in all situations. Then I use the tree under a roof example file, which I keep specially for testing (it’s included as a simplified testing file) to see how the component proliferates

I made an error when I was making the component the first time, but I left it in so you could see what it did, and how I fixed it. It comes from the fact that I drew my polygon anti clockwise when I made it, but vert grids draw them clockwise, and therefore the right hand rule means that the coordinate system pointed the wrong way.





Of course test rigs are useful in more cases that just for prototyping components.
Components are just a convenient illustration.
You can use this theory to test scripts with sample data, or just models.
The other issue is that you can set up a situation in a test that makes your model do something that it’d never be asked to do in real use. If it breaks at this point you get two options, ignore it because it’ll never get to that position, or fix it because you might want to reuse that function/component/model in the future. As long as you aren’t under fierce time pressure, the latter is always the better option.