# Chapter 11: Mr. DeMille, I'm Ready for my Close-up

## The Trouble with Toyotas

One of my friends recently moved from Santa Monica to Venice, California. While not a particularly lengthy move – only about a mile – she had to borrow a pickup truck, enlist friends (like me) and organize all of the specifics of the move. Her new home needed some minor repairs, so, on the day after the move, she rose early enough to be at the door of the local *Home Depot* when it opened, and spent an hour collecting all of the various hardware supplies she needed for the job, went through the check-out line, then made an exit to the parking lot, now full of cars.

I should mention that the car she'd borrowed, a white Toyota pickup truck, has proven very popular among the do-it-yourself set. So when my friend looked out into the parking lot, she saw a sea of white Toyota pickup trucks. And she'd forgotten exactly where she parked.

Why did this present such a problem? You probably know the answer very well; the features that make a car yours - the tiny dents and bumper stickers which cars acquire as they pass through this world – can't be seen unless you're close to the car. At a distance, two cars of the same color quite often look the same. Proximity marks the difference between my car and yours; as I get closer to my car, I can see the differences which makes my car mine.

## Waste Not, Want Not

So let's say that you're designing a palace in cyberspace – a place you can hang out with a few of your close friends. You spend months planing it, building it, and tuning it. It's huge – virtually and literally – because all of that work, all of that detail has given you a model that's too large to load on any but the most capable computers. You're stuck! You've created the ultimate virtual space, but it's too big for anyone to enjoy.

While all of this detail is absolutely necessary – to create the appropriate effect – it isn't always immediately necessary. In other words, when you're not looking at something, does it need to exist? More than an arcane question of Zen philosophy, this practical question that gets asked every day by anyone working in real-time computer graphics. In the virtual world every object that can be viewed must be painstakingly rendered by the computer; every object slows the computer down by just that much more. Too many objects and you'll rapidly mark the transition between real-time rendering and "slide-show" rendering, where things move so slowly you'll think you're watching a poorly-orchestrated slide show.

How can you make objects “go away” when you’re not looking at them? The answer lies in a concept known as *level-of-detail*.

# It’s Pronounced “Load”

There’s another good reason to use level-of-detail, one which has to do with the screens that we use when viewing virtual worlds. A very complicated object – a human figure, for example – looks like a blur of a few pixels when viewed at any distance. Unlike our eyes, our displays have relatively poor resolution; when objects pass a certain distance, it’s impossible to draw them in any detail. That wasted detail only slows the computer down unnecessarily. Why show it in all its glory? Wouldn’t it be just as good to substitute a dramatically simplified version of the object until you got close enough to make out some more details? Wouldn’t it be great if VRML had a node that could all of this for you?

It does. The LOD node (pronounced “L-O-D”, or “Load”) allows you to create several representations of an object, at differing levels of details, and tells the browser when to switch between these representations. Here’s the definition of the LOD node:

```
LOD {       # definition of LOD node
      level []          # MFNode, mult. values
      center            # SFVec3f
      range []          # MFFloat
}
```

Each of the representations of an object, from most complex to least complex, are provided as values in the level field, while the range field contains the list of *switch points* between the representations, from nearest to furthest. **There must always be one less value in the** range **field than there are representations in the** level **field.** In other words, two representations would require one range value, three representations, two range values, and so on. Anything else and – guaranteed – your browser will crash.

Most frequently, the representations in the level field are Shape nodes. Two representation, say of a Sphere and a Box, would require two Shape nodes. Here’s such an example, using an LOD node, with a switch point at five units (five meters in real-world terms):

```
#VRML V2.0 utf8
# This is the first example on level-of-detail
LOD {
      level [           # two representations
            # the most complex goes first
            Shape {           #  Create a visible shape
                  appearance Appearance {
                        material Material {
                              diffuseColor 0 1 0
                        }
                  }
                  geometry Sphere { }
            }
```

```
            # followed by the simplest
            Shape {           #  Create a visible shape
                  appearance Appearance {
                        material Material {
                              diffuseColor 1 0 0
                        }
                  }
                  geometry Box { }
            }
      ] # end list of representations
      range [ 5 ] # switch-point at five units
}
```

When the browser loads the world, we see the Box, as we are positioned more than five units away from the object. But if we move toward the object, we can see that it magically turns into the Sphere, once we cross the five-unit threshold.

As you move back and forth across that threshold, you'll see the object change its form back and forth, from the simple Box, to the complex Sphere. If we wanted to cycle between a Box, a Cylinder, and a Sphere, with switch points at eight and four units, it might look like this:

```
#VRML V2.0 utf8
# This is the second example on level-of-detail
LOD {
      level [           # three representations
            # the most complicated goes first
            Shape {           #  Create a visible shape
                  appearance Appearance {
                        material Material {
                              diffuseColor 0 1 0
                        }
                  }
                  geometry Sphere { }
            }
            # followed by the less complicated
            Shape {           #  Create a visible shape
                  appearance Appearance {
                        material Material {
                              diffuseColor 1 0 1
                        }
                  }
                  geometry Cylinder { }
            }
            # followed by the simplest
            Shape {           #  Create a visible shape
                  appearance Appearance {
                        material Material {
                              diffuseColor 1 0 0
                        }
                  }
                  geometry Box { }
            }
      ] # end list of representations
      range [ 4, 8 ] # switch-points at 4 and 8 units
```

```
}
```

When we open this in a browser, we see the Cylinder, because it's come up in the middle of the range.

But if we move out, we can see that it's turned into a Box:

And as we move it, it cycles first to the Cylinder, and then finally to the Sphere:

# Getting Real

While the preceding examples help to illustrate the functionality of the LOD node, neither of them portray the real power of the node; for this, we'd need to have an example that we might find in a real-world situation. Let's say that we want to create a realistic model of the Earth; we did this in our last chapter, using texture maps. When we're far away from the model - say 200 units – we'd want to show it as a cyan Box, and, as we got closer, transition to a cyan Sphere, then, finally, show it in all of its texture mapped glory. Here's how we might do that:

```
#VRML V2.0 utf8
# This is the third example on level-of-detail
LOD {
      level [           # three representations
            # the most complicated goes first
            Shape {           #  Create a visible shape
                  appearance Appearance {
                        texture ImageTexture {
                              url [ "worldmap.jpg" ]
                        }
                  }
                  geometry Sphere { }
            }
            # followed by the less complicated
            Shape {           #  Create a visible shape
                  appearance Appearance {
                        material Material {
                              diffuseColor 0.5 1 1
                        }
                  }
                  geometry Sphere { }
            }
            # followed by the simplest
            Shape {           #  Create a visible shape
                  appearance Appearance {
                        material Material {
                              diffuseColor 0.5 1 1
                        }
                  }
                  geometry Box { }
            }
      ] # end list of representations
      range [ 50, 200 ] # switch at 50 and 200 units
}
```

The world will load at its closest resolution, with the texture-mapped Sphere.

Then as we back away, it transitions to a similarly-colored Sphere.

Then – as we get too far away to notice or care – it turns into a Box of the same color.

Why is this important? Performance, performance, performance. The Box will render – on average – fifty times faster than the Sphere, and perhaps a hundred times faster than the texture-mapped Sphere. Let's say that you were modeling a solar system, or a galaxy – and someone has done this in VRML – you could show fifty times as many stars if you used Box nodes to represent them than if you used Sphere nodes. However, you could have the Box automatically change into a Sphere as the user grew closer to any of them – having it both ways, virtually.

## Great Savings, Now and Later!

There's yet another reason to use the LOD node, but we won't learn about inline objects until the chapter after next. But I can tell something now – not only can LOD make your worlds easier to render, it can also make them quicker to load – and that's always a user's first concern. More on this in chapter 13.

For now, let's turn our eyes back to the Web – after all, VRML is a Web technology, and learn how we can link our worlds into the Web…