# EXPLORING NORMAL VECTORS IN THE SECOND LIFE METAVERSE

Przemyslaw Bogacki
Department of Mathematics and Statistics
Old Dominion University
Norfolk, VA 23529
pbogacki@odu.edu

## *Introduction*

This article is based upon a handout distributed to participants of the ICTCM minicourse "Second Life: A Hands-on Introduction for Mathematics Professors". While the minicourse contained a number of modules and activities, here we focus only on a relatively small portion of these: we shall illustrate how a combination of basic Second Life building and scripting tools can be used to illustrate normal vectors.

In order to follow this article, the reader should have a basic familiarity with the Second Life Viewer 1.23 [1] (viewer 2 has been officially released after this minicourse, and is not discussed here).

## *LSL scripting*

The scripting language used in Second Life is called Linden Scripting Language, or LSL. Its syntax resembles many other languages, so a reader familiar with one of the modern computer programming languages should be able to recognize a number of constructs used in a typical LSL code.

Let us create a new box on the ground, then (with the box selected for editing), go to the "Content" tab, and click on the "New Script" button. Under the "Contents" folder a new item will appear, labeled "New Script". Double-click on it to open it in the script editor.

*Figure 1.*
*The default LSL script.*

```
default
{
 state_entry()
 {
  llSay(0,"Hello, Avatar!");
 }

 touch_start(integer total_number)
 {
  llSay(0,"Touched.");
 }
}
```

This is an exceedingly simple program that is intended to do two things: say "Hello, Avatar!" when the script starts, and say "Touched." whenever the object containing this script is being touched by someone. A script will generally call some built-in functions, all of which have names that start with "ll". Our simple script only uses the function llSay with two parameters: the channel number (0 is the public channel that can be "heard" by everyone) followed by the text of the message. There are a couple of hundred built-in functions intended to facilitate user-object interaction, object motion (e.g., vehicles can be constructed), altering object's shape and texture. Information about these functions, as well as

other aspects of the LSL syntax (data types, flow control, etc.) can be found on many web sites, including the official Second Life wiki [2].

***Displaying the normal vector components***

Recall from the previous section that when the default script is running inside an object, it will say "Touched" whenever anyone touches the object. In this section, we shall get the script to display the components of the unit normal vector to the surface of the object where it was touched.

With a default box (create one if needed) selected for editing, go to the "Content" tab. If the default script "New Script" has not been created yet, do so now by clicking on the "New Script" button. Now edit the script so that it matches the one shown in Figure 2.

*Figure 2. This LSL script displays components of the unit normal vector at the point where an object containing the script was touched.*

```
default
{
 state_entry()
 {
 }

 touch_start(integer total_number)
 {
  llSay(0, "Normal vector: "+(string)llDetectedTouchNormal(0));
 }
}
```

The function llDetectedTouchNormal(n) returns the unit normal vector where the object is touched (the integer parameter 0 corresponds to the first user touching the object in case if there is more than one.) The phrase "(string)" in front of the function name performs the typecasting of this vector, converting it to a string that can be displayed. Finally, the "+" between the strings is used to concatenate them.

Hit the "Save" button under at the bottom of the script editing window, and wait until the status messages say

Compile successful!
Save complete.

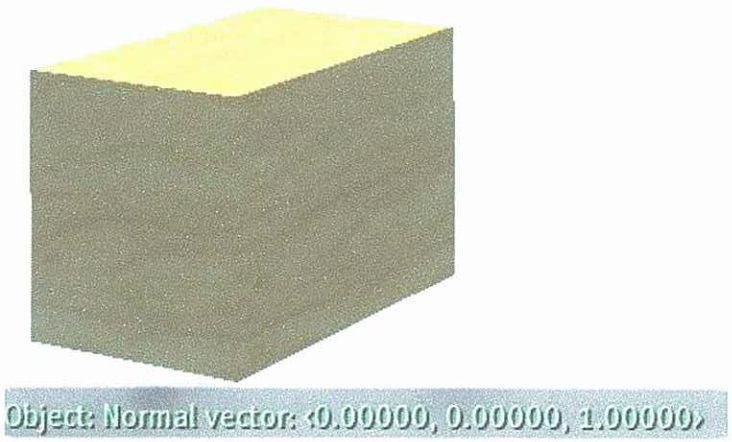
(If you receive error messages instead, you will need to make the necessary corrections, then proceed to save again.)

Close the script editing window and the tool window, then try touching different faces of the box. (E.g., touching the top should result in a message shown in Figure 3.)

After you are reasonably certain the new script is working as intended, select the box for editing again and go to the "Content" tab. Right-click on the "New script" and choose "Rename" from the pop-up menu. Change the script name to "SayNormalCoordinates". Now open your Inventory window, and drag the script you just renamed to the folder "Scripts" in your Inventory.

Next, let's create a sphere on the ground, open it for editing, and open the "Content" tab. In your Inventory window, expand the "Scripts" folder, and drag the "SayNormalCoordinates" script from it to the "Contents" folder in the sphere's tool window. Close the tool window, and try touching the sphere a few times.

*Figure 3.*
*When the top face of the box containing the script from Figure 2 is touched, a message reporting the components of the normal vector <0,0,1> is displayed.*




***Creating an arrow by linking two prims***

Vectors in 3-space are often visualized as arrows. While there is no Second Life arrow "prim" (or primitive shape), in this section we are going to construct an arrow from other shapes. Most objects in Second Life consist of multiple prims linked together – doing so allows the linked object to be moved, rotated, etc. without having to worry about maintaining the relative positions if individual prims.

We will build an arrow to represent a unit vector (1 meter long) that will be made up of two connected prims: a cylindrical stem, and a conical tip (see Figure 4(a)).

To **build the stem**, create a cylinder on the ground, then proceed to edit the following parameters:

**Size: X = 0.05, Y = 0.05, Z = 1.6; Slice B(egin) = 0.5**. Change the **color** to **red**, and the **texture** to **blank**.

You may notice that we are seemingly making the vector too long since Z=1.6m exceeds the desired length of 1m, however, the slice cut will trim one half of this length off, leaving the stem at 0.8m. This is done so that the stem's center will actually be at the initial point of the vector, which will make it easier to position the vector later on.

Proceed to **build the tip**: create another cylinder on the ground, then edit it as follows:

**Size: X = 0.1, Y = 0.1, Z = 0.2; Taper X = 1, Y = 1**. Change **color** to **blue**, and **texture** to **blank**.

Before the two prims are linked, they should be properly **aligned**. An easy way to do that is by using the world grid. Begin by moving the red stem to a location where all three coordinates: x, y, and z are **integers** (see Figure 4(b)). You can do that by using the "snap-to-grid" feature when dragging the prim along an axis (drag the mouse pointer toward the white grid to have the position snap to the grid). As a quicker (if somewhat riskier) alternative, you can enter integer coordinates directly in the floating tool window, however, make sure to correctly enter values **very close to the current position** – otherwise your object may end up drifting out of sight. (A good way to keep the object close is by rounding X and Y up or down to the nearest integer; Z should always be rounded up so that the object does not hide under ground level.)

Now, move the tip to the same X and Y position as the stem, but with Z position 0.9 higher (Fig. 4(c)).

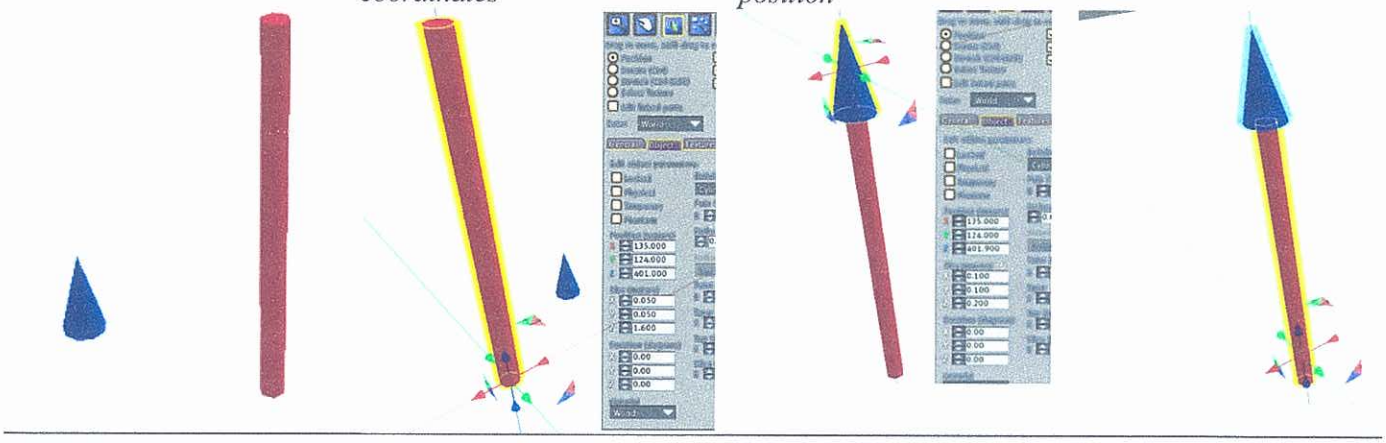
*Figure 4. Building a vector by linking a cylinder and a cone.*

*(a) The tip and the stem*

*(b) Positioning the stem at a point with integer X, Y, and Z coordinates*

*(c) Positioning the tip 0.9m above the stem position*

*(d) Linking the tip to the stem*



Visually verify that the two prims are properly aligned.

It is now time to **link** the two prims (see Figure 4(d)). Follow these steps:

**a.** Select the blue tip for editing (a yellow highlight should now surround the tip)
**b.** Holding the <Shift> key, left-click on the red stem (both the tip and the stem should now be highlighted in yellow).
**c.** From the "Tools" menu at the top of the Second Life window, select "Link" (the stem should still be highlighted in yellow, but the tip highlight color should change to light blue).

Try moving the object now – you will see that both prims are now moving *in sync*. Set the object's name under the floating tool window "General" tab to "Unit vector" then take it to your inventory (right-click on the object and select "Take").

***Showing normal vector as an arrow***

Let us now use the arrow we created in the previous section to provide a visualization for unit normal vectors. With a default 0.5m x 0.5m x 0.5m box selected for editing, and its "Content" tab selected:

i. drag the "Unit vector" from your inventory into the prim's "Contents" folder;
ii. edit the prim's script to match the contents of Figure 5.

*Figure 5. The script that displays an arrow representing a normal vector at the point where the surface of the prim has been touched*

```
default
{
 touch_start(integer total_number)
 {
  vector N=llDetectedTouchNormal(0));
  vector B=llDetectedTouchBinormal(0));
  llRezAtRoot("Unit vector", llDetectedTouchPos(0),
   ZERO_VECTOR, llAxes2Rot(B,N%B,N), 0);
 }
}
```

Here are a few comments explaining the statements of this script:

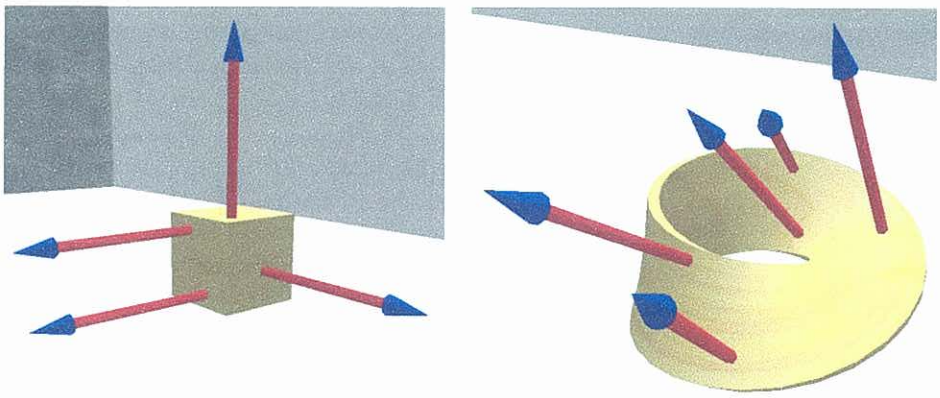
- The function llAxes2Rot(x1,y1,z1) represents a rotation of the object's original axes x,y,z to the orthonormal vectors x1, y1, z1, respectively.
- N%B calculates a cross product of the normal and binormal vectors.
- llDetectedTouchPos(0) returns the position (in region coordinates) where the object has been touched.
- llRezAtRoot will create ("rez") a copy of the arrow with the desired location and direction.

Close the tool window, and touch a few different faces of the box.

Try repeating steps i. and ii. for some of other prim shapes (e.g., a torus in Figure 6).

---

*Figure 6.*
*After the "Unit vector" object and the script of Figure 5 have been added to a prim's inventory, touching the prim results in displaying arrows representing the normal vectors.*



---

### *Conclusions*

The ability to illustrate normal vectors inside Second Life could potentially be used in a number of contexts. In multivariable calculus alone, this topic comes up repeatedly when discussing a variety of topics such as gradient, Lagrange multipliers, flux integral, divergence theorem, and Stokes' theorem. While this could be useful in classroom demonstrations or student hands-on (in world) explorations, it is just a small example, which serves to demonstrate some of the potential that virtual worlds such as Second Life have for undergraduate mathematics courses.

### *References*


[1] Second Life User's Manual, URL: http://wiki.secondlife.com/wiki/User%27s_Manual

[2] LSL (Linden Scripting Language) Portal, URL: http://wiki.secondlife.com/wiki/LSL_Portal