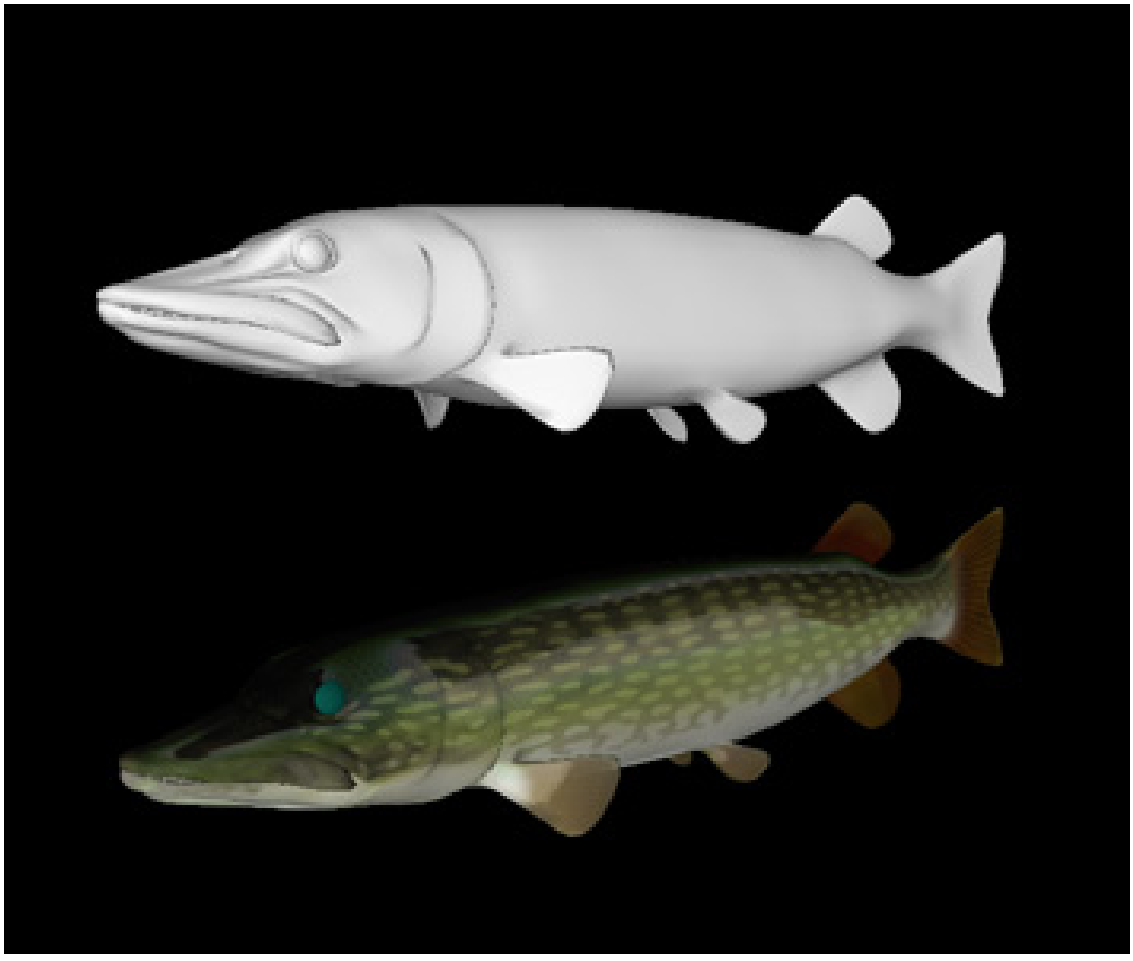
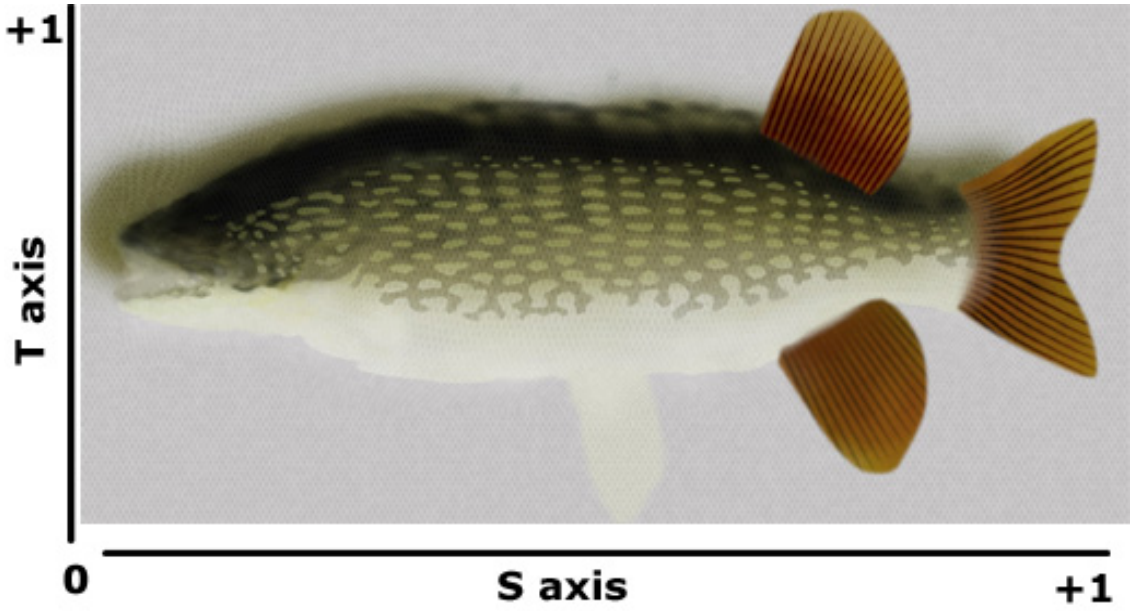


TEXTURE COORDINATES

Textures are usually 2D image files (e.g. jpegs or tif files). In order for these 2D files to appear in 3D space, they need a surface to **wrap** onto. (In the examples in this section we'll be wrapping textures on texture quads which are simply a plane in 3D space.)

To determine how 2D files should be wrapped on 3D surfaces, we use **texture coordinates**. The texture coordinates of a 3D surface define where any texture should be applied to that surface. Texture coordinates have two axes: the S axis which refers to the width of an image, and the T axis which refers to the height. Regardless of texture or model scale, texture coordinates range from 0 to 1.0 in both dimensions with (0, 0) at the bottom left.



Each point on the surface of a 3D model is assigned an S,T - coordinate so that when a texture is wrapped onto that model, the color of the pixel at point S,T on the texture will be used to color that point on the model. For example, let's say I have a 3D model of a head and the S,T - coordinate assigned to the tip of the nose is (.5,.5). When I wrap a texture on that model, the model will take the color at point S,T on the texture and use that color to paint the tip of the nose.

In order to apply a texture to a model, the model needs S,T coordinates. Modeling software (e.g. 3DSMax) calls the process of wrapping geometry in textures S-T mapping or U-V mapping.

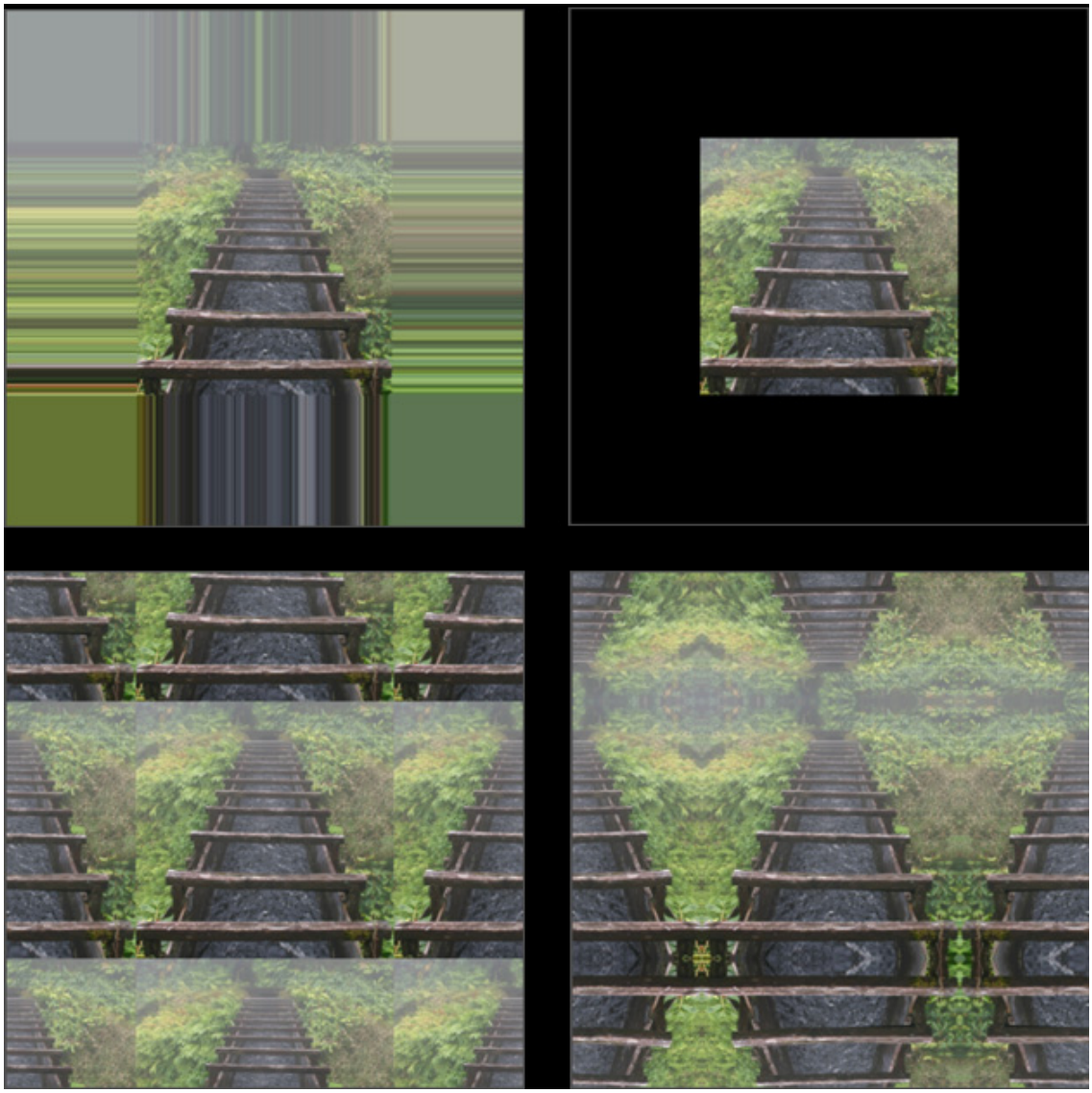
To get a feel for how texture coordinates work, run the texture coordinates demo. In this program, a texture has been applied to a texture quad (a 3D model that's simply a 3D plane). When the program first loads, the texture quad's texture coordinates are assigned such that a square texture maps perfectly onto the quad. The 0,0 point on an image falls on the bottom left of the quad, the 1,1 point on the image falls on the upper right of the quad, and etc. In this demo, however, you can alter the quad's texture coordinates by changing their scale or by translating their position across the surface. Try going to the "texture coordinates" menu and changing the scale of S and T coordinates to "2". This change will double the texture coordinates in both dimensions and, as a consequence, the image will only fill half of the surface in each dimension. (At first it might seem counterintuitive that doubling texture coordinates would shrink the image, but remember that the surface's texture coordinates are what we're

doubling, not the image size. So, the point that was originally assigned to (.5,.5) in the image is now assigned to (1,1) in the image.)

Along the same lines, you can also change the position of the model's texture coordinates by translating them. Try changing translating the texture coordinates of the model by setting the translation of both S and T coordinates to "-.5. This translation should move the image up and to the right. (If this again seems counterintuitive, consider that the point on the surface assigned to (1,1) will be assigned to (.5,.5) after the translation of the surface's S-T coordinates

As you can see in this demo, texture coordinates can be assigned in such a way that the texture does not cover the entire object. **Texture wrap modes** specify how the rest of the object (where the texture coordinates fall outside of the 0 to 1 range) should be textured. One option for wrapping is to repeat the same texture over the surface of the object. Mirroring also repeats the texture, although with mirroring the texture is reflected for each iteration across the given dimension. If you don't want to repeat a texture, you can **clamp** the texture and use either the border color or the pixels at the very edge of the texture to color the rest of the object.

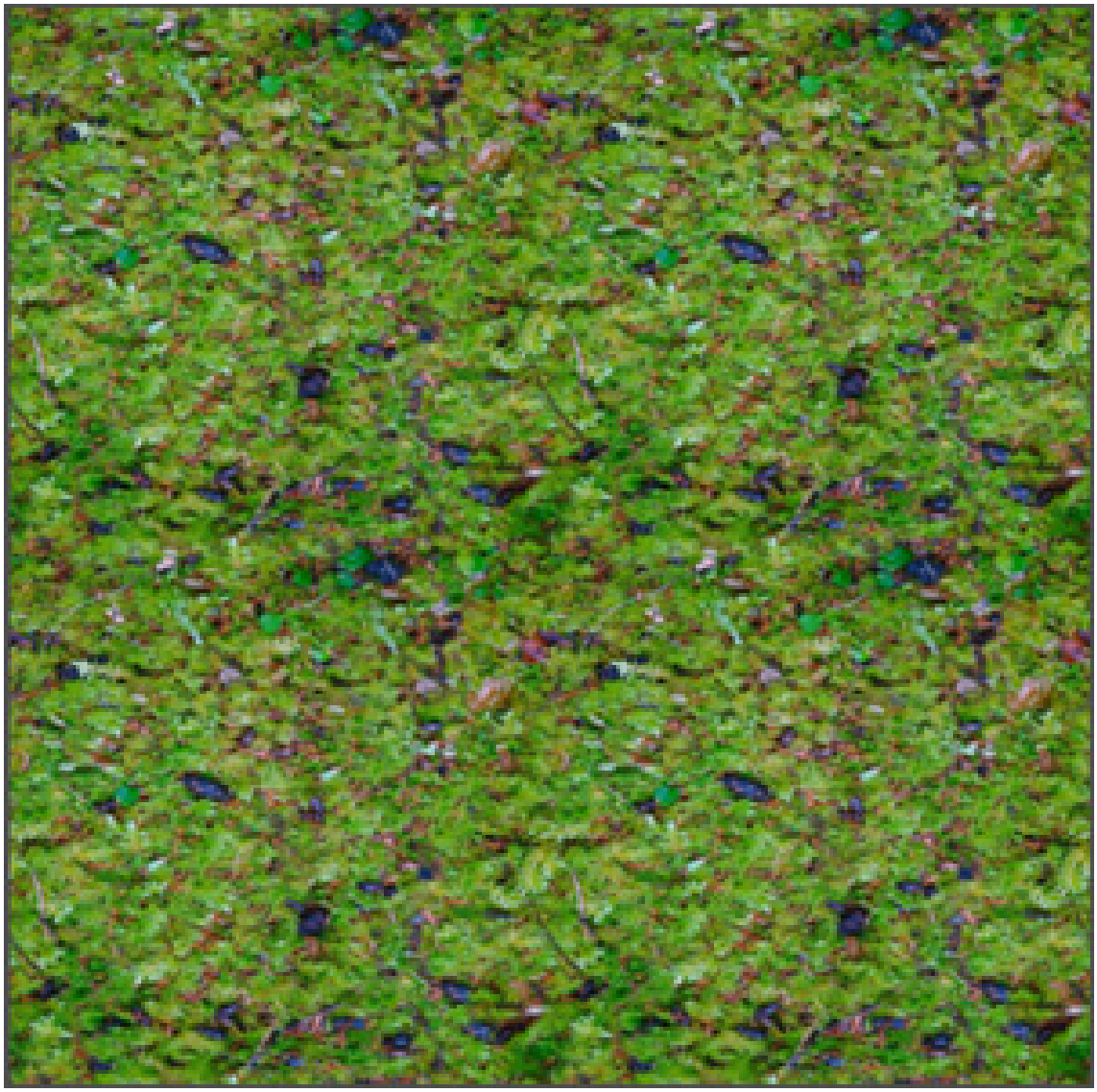
Run the texture coordinates demo and after scaling the texture coordinates so that the texture does not fill the entire quad, go under "wrap modes" and play with the different options. If you clamp to the border, try changing the border color with the "border color" menu.



When it comes to repeating textures across the surface of a model, it's important to think about whether or not the texture you're using is **tileable**. A texture is tileable when it's similar enough on it's edges that when you place a number of copies of that texture next to each other, it's difficult to tell where one ends and the other one begins.

Run the texture coordinates demo and again scale the texture coordinates to that the texture does not fit the quad. Set the wrap mode to "REPEAT" and go to the "blend" menu and move the bar all the way to the right. Notice how it's hard to distinguish

where this texture begins and ends. A texture like this one would work well on a large model with a lot of tiling, such as a model for the ground.



MULTITEXTURING

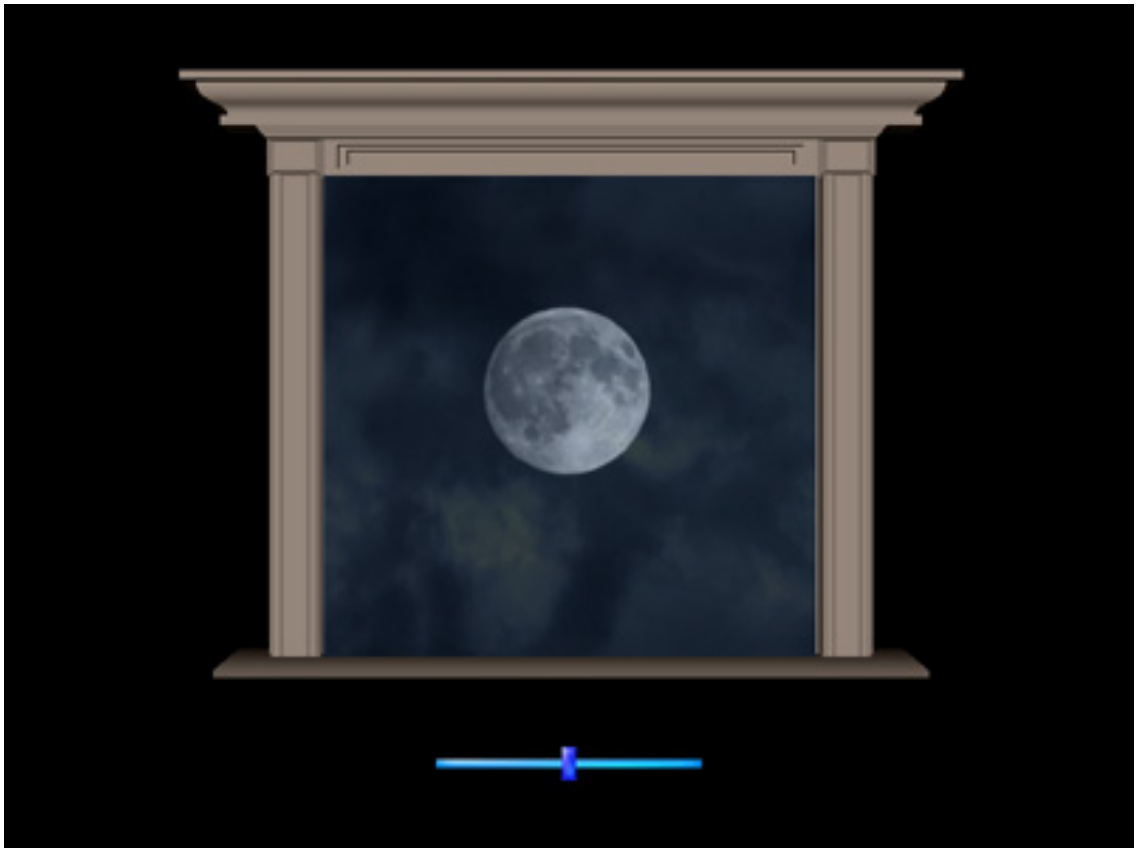
With **multitexturing**, surfaces can have more than one texture. Multitexturing can give a surface a more complex appearance by adding light patterns or the simulation of surface dynamics. You can also blend between textures to simulate a changing

appearance (such as a leaf turning color). 3D modeling software products have a host of options for applying multiple textures and defining the relationships between them (bump-maps, alpha maps, etc.).

The "blend" menu in the texture coordinates demo made use of multitexturing. As you move from the original texture to the more tileable texture with the sliding bar, you're changing the degree to which each texture is being used to color the texture quad.

EXAMPLE SCRIPT: APPLYING TEXTURES

In this example script, we will take a piece of a model and then apply a couple textures to it, and manipulate the texture coordinates for those textures. Open your own script for this example, and as we go through pieces of code, add them to your script to see their effects.



First we import the viz library and begin rendering a scene with the go command. After that, we take the main viewpoint and place it such that we'll be able to look at the scene when it loads. Next we add a model to the scene and grab one of it's chil-

dren, a **sub-node** called "glass". Both of these commands will return node3d objects (that we've labelled "model" and "window") that we can use to control these models.

```
import viz
viz.go()

#Put the viewpoint in a good place to
#see the texture quad.
viz.MainView.setPosition( 0, .68,-1.3 )
#Add a model and grab one of its
#children to put the textures on.
model = viz.add('art/window.ive')
window = model.getChild( 'glass' )
```

Next we'll add the texture files with the viz library's "addTexture" command. These two files are jpegs in our art directory.

```
#Add the texture files.
clouds = viz.addTexture('art/tileclouds.jpg')
moon = viz.addTexture('art/full moon.jpg')
```

Now we'll apply these textures to our window with the node3D "texture" command. This command allows you to add a given texture to the model and allows you to specify which unit the texture will go in. Specifying the unit is important when you're applying multiple textures to a model.

```
#Apply the textures to the window.
#The moon will go in the window's
#first unit(0) and the clouds will
#go in the second (1).
window.texture( clouds, '', 1 )
window.texture( moon, '', 0 )
```

To control the ways in which the cloud texture wraps, we'll specify a wrap mode with the wrap command which accepts two flags-- one for the dimension (S or T) and the other for the wrap mode itself. Here we'll use the repeating texture mode.

```
#Set the wrapping mode for the clouds
#to be REPEAT so that as the surface's
#texture matrix translates, there will
#still be texture to show.
clouds.wrap( viz.WRAP_S, viz.REPEAT )
clouds.wrap( viz.WRAP_T, viz.REPEAT )
```

Next we're going add a slider device to the scene so that the user can interact with it. Specifically, the user will be able to push the slider back and forth to control the blend

of the two textures on the model. First we add a slider object (a node3D gui object) and set its position on the screen. Then we add a function that will be called every time the user changes the slider. Codewise, we're setting up a slide event here with the onslider command from the vizact library. When the slider event occurs, it will call our swap_textures function and tell that function where the slider has been moved to. Our swap_textures function will take the slider's position and blend in the clouds (the texture in unit 1 of the model) anywhere from 0 (none) to 1 (the object is 100% textured by the clouds texture).

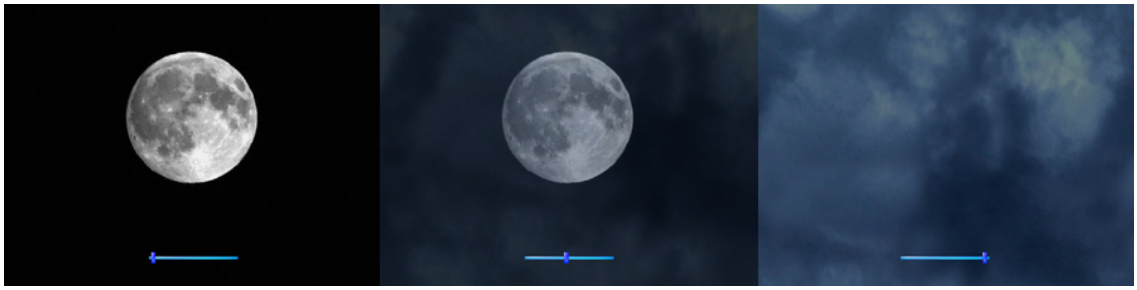
```
#Add a slider and put it on
#the bottom of the screen.
slider = viz.addSlider()
slider.setPosition(.5,.1)
#This function will be called
#every time the slider
#is moved and will swap the
#textures according to the
#slider's position.
def swap_textures( slider_position ):
   #Use the slider's position to get
   #the amount of cloud blend.
   cloud_amt =  slider_position
   #Blend the clouds (unit #1) in that amount.
   window.texblend( cloud_amt, '', 1 )
#Set up the slider event to call our function.
vizact.onslider( slider, swap_textures )
#Set the initial blend to match the slider.
swap_textures(0)
```

Run the script now to see how that looks. Try playing with the slider and backing up so that you can see the entire model.

Next we're going to manipulate the window's texture coordinates to move the clouds across the surface. To do this, we first create a transform matrix using the vizmat library. This matrix will include all the information about our transform of the texture coordinates including the scale and translation. Then we'll use a timer to call the move_clouds command (timers are discussed in more detail in the Modeling action section). The move_clouds command applies a translation to our transform matrix with the "postTrans" command. It then applies that matrix to unit 1 (the cloud texture) of our window's texture coordinates with the node3d texmat command.

```
#Create a matrix that we'll use to
#the surface's texture matrix.
matrix = vizmat.Transform()
#This function will move the clouds incrementally.
```

```
def move_clouds():
   #Post translate the matrix (move it in the s and t dimensions).
   matrix.postTrans(.0005,.0005,0)
   #Apply it to the surface.
   window.texmat( matrix, '', 1 )
#Run the timer every hundredth of a second.
vizact.ontimer( .01, move_clouds )
```



Now run the script again and you should see the clouds moving when you slide the slider to the right.

EXERCISES

1. Add the "art/wall.ive" model to a world, add the "art/stone wall.jpg" texture and apply it to that model.

2. Change the scale of box's texture coordinates for the unit that the stone wall texture is in. Then repeatedly tile the texture across the box.

3. Add the "art/pine needles.jpg" to another unit of the wall model from the previous exercise and blend the two textures so that the wall looks like the picture below.