Tutorial 7: How to Script Scene Object Transformations Using the New vrdTransformNode Introduced in VRED 2021.3

The *Tutorial 7: How to Script Scene Object Transformations Using the New vrdTransformNode Introduced in VRED 2021.3* video covers examples with cameras and viewpoints, as well as sceneplates in VR.

Download the example scripts here: https://github.com/raemorris/VRED-Python-Tutorial-Examples-Scripts/blob/main/tutorial7.zip?raw=true

Sample Python Code

Here are the accompanying example Python scripts for the *Tutorial 7: How to Script Scene Object Transformations Using the New vrdTransformNode Introduced in VRED 2021.3* video.

- To view the video, visit: https://www.youtube.com/watch?v=1Vu4Ugwsclc
- To review the video caption, see *Video Caption*.

old_vs_new_transformations.py

```
# Old API transformations


# Create a point p1
x1 = -500
y1 = 0
z1 = 1500


p1 = vrNodeUtils.createSphere(1, 100, 1, 1, 1)
p1.`setTranslation`(x1, y1, z1)
p1.setName("p1")


# Create a point p2
x2 = 500
y2 = 0
```

```
z2 = 500

p2 = vrNodeUtils.createSphere(1, 100, 1, 1, 1)
p2.`setTranslation`(x2, y2, z2)
p2.setName("p2")

# Create a point p_center at the geometrical center between p1 and p2
x_c = (x1 + x2) / 2
y_c = (y1 + y2) / 2
z_c = (z1 + z2) / 2

p_center = vrNodeUtils.createSphere(1, 100, 1, 1, 1)
p_center.`setTranslation`(x_c, y_c, z_c)
p_center.setName("p_center")



# NEW API transformations

# Create a point p1
v1 = QVector3D(-500, 500, 1500)
p1 = vrNodeUtils.createSphere(1, 100, 1, 1, 1)
vrNodeService.getNodeFromId(p1.getID()).`setTranslation`(v1)
p1.setName("p1")

# Create a point p2
v2 = QVector3D(500, 500, 500)
p2 = vrNodeUtils.createSphere(1, 100, 1, 1, 1)
vrNodeService.getNodeFromId(p2.getID()).`setTranslation`(v2)
p2.setName("p2")

# Create a point p_center at the geometrical center between p1 and p2
v_center = (v1 + v2) / 2
```

```
p_center = vrNodeUtils.createSphere(1, 100, 1, 1, 1)
vrNodeService.getNodeFromId(p_center.getID()).`setTranslation`(v_center)
p_center.setName("p_center")
```

vrdTransformNode.py

```
groupNode = vrNodeService.findNode("Group")
deleteNode(groupNode)

numberOfBoxes = 50
boxSize = 50
xStep = 125
xRotationStep = 20
z = 500
radius = 400

boxGroup = createNode("Group")

for i in range(0, numberOfBoxes):
    box = vrNodeUtils.createBox(boxSize, boxSize, boxSize, 1, 1, 1, 0, 0, 0)
    box.setName("box {}".format(i))
    boxGroup.addChild(box)

    # Get the vrdNode object of the box
    boxNode = vrNodeService.getNodeFromId(box.getID())

    boxTranslation = QVector3D(xStep * i, 0 ,z)
    boxNode.`setTranslation`(boxTranslation)

    boxScale = boxNode.getScale() * (1 + (i * 0.01))
    boxNode.setScale(boxScale)
```

```
boxLocalRotation = QVector3D(xRotationStep, 0, 0) * i
boxNode.`setRotation`AsEuler(boxLocalRotation)

boxNode.`setWorldRotatePivot`(QVector3D(0, 0, 0))
boxNode.`setRotatePivotTranslation`(QVector3D(0, 0, z + radius))
```

Video Caption

Hello and welcome to a Python Tutorials for VRED Pro. I'm your host, Christopher, and today, I want to show you how we can use the new vrdTransformNode to script object transformations.

Up until the recent release of VRED version 13.3, the API version 2 did not include the vrdTransformNode class we are talking about in this video. The vrdTransformNode class combines all transformation operations that can manipulate geometry nodes. In the previous VRED versions, you would have done transformations with the functions provided in the vrNodePtr class, like setTranslation, setRotation, and so on. The functionality of the new vrdTransformNode itself hasn't changed that much compared to the old vrNodePtr. But the way we work with these functions has.

When we have a look at the old transform operations, we can see that we have to pass all of our parameters as single values. So, we pass the x, y, and z component of a vector separately into the function. And, return values are also a list of vector components.

The current example shows a simple script that creates two points and calculates a center point between them. We have to handle each vector component separately and pass them into the translate function. This is all fine, until we want to do some vector maths, like calculating the center point between two positions. We would have to convert these values in a proper vector representation or write our own math function to do this. Here, we just calculate the values by hand.

To solve this problem, all the transform operations are now part of the vrdTransformNode and use the new "Qt data types" The new data types combine the separate x, y, and z components to a single vector and gives us the ability to directly use them in vector maths. Looking at the same example as before, we can see that we do not have to the calculate the center point by hand. We can directly add the two vectors together, divide by two, and use the resulting vector as the input of our translate function. So, we do not have to import external libraries or write our own "vector operations". This makes the code far more readable and maintainable.

Let's have a look at another example. I want to generate a bunch of boxes that are spiraling along a line. They should be rotated against each other and change their scale, depending on their position. This is a nice example to show how some of the transform operations are working. Of course, we could do this with the old API but the new "transform node" really helps us with the math.

Let's start with creating a group node where we want our boxes to be. We define a few variables that will control the behavior of our spiral of boxes. The next step is to create our boxes in a loop, add them

to the group node, and move them along the x-axis. The createBox function returns a “node pointer” from the old API, so we have to convert it to a vrdTransformNode with the getNodeFromId function. Then, we just create a simple vector that holds our translation and put it in the setTranslation function of our box. The “I” parameter from our loop is responsible for shifting each box a little bit further along the x-axis.

Next, we want to change the scaling of our boxes along the x-axis. To do this, we first get the current scaling of our box and multiply it by a small factor that increases with the loop increment. We use the resulting vector as an input parameter of the setScale and see that the boxes are now getting larger.

As a last step, we want to spiral our boxes around a center line. And, for this, you have to trust me a little bit. At first, we create a rotation vector that holds the euler angles for our rotation and set the rotation of our box. This way, each box has a nice rotation to it. Now, we call the setWorldRotatePivot function with a zero vector and the setRotatePivotTranslation with a vector that specifies the position of our rotation pivot. What it does it basically moves the rotation axis away from our boxes so that the rotation transform gets shifted and produces this nice spiral effect. The rotation no longer is applied at the center of our boxes but on an axis above our boxes.

As I already said, we could have achieved the same result with the old API, but the new “transform node” makes it much easier. You can find the documentation on these new data types when searching for “Python QT vector” in the “qt framework documentation”. For example, in the Qt vector documentation, you can find all mathematical functions you can do with this data type, like the base operations addition and subtraction, or functions like normalizing the vector. We also have another tutorial where I talk about the new data type in VRED more in depth.

That’s it for today! I hope you enjoyed it. Thanks for joining and see you for our next tutorial.